IMPLEMENTASI *DYNAMIC DIFFICULTY ADJUSTMENT* PADA *RACING GAME* MENGGUNAKAN METODE *BEHAVIOUR TREE*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Isthofi Aslim Sofyan
NIM: 115060807111060

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

Implementasi Dynamic Difficulty Adjustment Pada Racing Game Menggunakan Metode Behaviour Tree

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Isthofi Aslim Sofyan
NIM: 115060807111060

Skripsi ini telah diuji dan dinyatakan lulus pada
03 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Muhammad Aminul Akbar, S.Kom., M.T	Tri Afirianto, S.T, M.T
NIK: 20160789 1013 1 001	NIK: 201309 851213 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

IDENTITAS TIM PENGUJI

Penguji 1

Nama : Eriq Muhammad Adams Jonemaro, S.T, M.Kom

NIP/NIK : 19850410 201212 1 001

Penguji 2

Nama : Wibisono Sukmo Wardhono, S.T, M.T

NIP/NIK : 201008 820404 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 06 Agustus 2018

Isthofi Aslim Sofyan

NIM: 115060807111060

DAFTAR RIWAYAT HIDUP

Nama : Isthofi Aslim Sofyan

Tempat Tanggal Lahir : Malang, 10 Mei 1994

Riwayat Pendidikan : TK Al – Aqsha

SDN Harapan Jaya IX

SMPIT YAPIDH

SMAI Darussalam

KATA PENGANTAR

Puji syukur kehadirat Allah SWT karena atas rahmat dan hidayah-Nya, penulis dapat menyelesaikan skripsi dengan judul "**IMPLEMENTASI *DYNAMIC DIFFICULTY ADJUSTMENT* PADA *RACING GAME* MENGGUNAKAN METODE *BEHAVIOUR TREE***".

Penyusunan skripsi ini tidak lepas dari bantuan semua pihak yang telah memberikan semangat, doa, bimbingan, kritik, serta saran. Maka dari itu penulis menyampaikan ucapan terimakasih kepada:

1. Bapak Aminul Akbar, S.Kom., M.T dan Bapak Tri Afirianto, S.T, M.T sebagai dosen pembimbing 1 dan dosen pembimbing 2 yang telah memberikan ilmu, masukkan, dan saran yang bermanfaat dalam proses penyelesaian skripsi ini.
2. Rekan-rekan yang telah memberikan dukungan baik secara moral, pemikiran maupun doa.
3. Semua pihak yang tidak dapat penulis sebutkan satu per satu yang terlibat baik secara langsung maupun tidak langsung demi terselesaikannya tugas akhir ini.

Semoga skripsi ini dapat bermanfaat dan berguna bagi pembaca terutama mahasiswa Prodi Teknik Informatika Jurusan Teknik Informatika Universitas Brawijaya.

Malang, 06 Agustus 2018

Penulis

isthofiaslim@gmail.com

ABSTRAK

Video games merupakan hiburan dan tantangan. Tanpa tantangan *video games* akan mudah diselesaikan dan membosankan. Namun apabila tantangan terlalu sulit dapat membuat pemain frustasi dan menyerah. Hal ini berhubungan dengan *flow-state* yaitu ketika kemampuan pemain dan tantangan dari *game* setara. Pada umumnya setiap *game* menyediakan pengaturan tingkat kesulitan. Tingkat kesulitan yang disediakan biasanya disediakan dalam bentuk pilihan dari tingkat kesulitan mudah *(Easy),* sedang *(Medium),* dan sulit *(Hard).* Sayangnya model pengaturan seperti ini bersifat statis sehingga menimbulkan ketidaksetaraan antara pemain dan tantangan dari *game*. Untuk menyelesaikan masalah tersebut *dynamic difficulty adjustment* (DDA) diterapkan dalam penelitian ini. DDA adalah alternatif dari pengatur tingkat kesulitan statis yang harus ditentukan oleh pemain sebelum memulai permainan, dengan adanya DDA pemain tidak perlu repot mengatur tingkat kesulitan sebelum bermain. DDA berfungsi sebagai pengatur tingkat kesulitan yang bekerja secara otomatis berdasarkan kemampuan pemain. Untuk mendukung penerapan DDA, *behaviour tree* digunakan untuk membantu proses adaptasi AI terhapadap kemampuan pemain. Pengujian dilakukan dengan uji coba balap sebanyak 3 *lap* dan dicatat nilai jarak tempuh dari setiap checkpoint yang dilewati. Nilai jarak dari masing – masing uji akan dijumlahkan dan dihitung rata-rata selisih jarak. Dari pengujian yang dilakukan didapatkan hasil penerapan *behaviour tree* dan DDA menghasilkan permaninan yang tidak membosankan dan tidak terlalu sulit untuk pemain. *Behaviour tree* dan DDA menghasilkan kemampuan AI lawan yang dinamis dan mampu menyesuaikan kemampuan dari pengguna.

Kata kunci: tingkat kesulitan dinamis, DDA, *behaviour tree*, *racing game*.

ABSTRACT

Video games is an entertainment with challenge. Video games would be too easy and boring without challenge. But if the challenge is too hard that would frustrate many people and made them gave it up completely. This related to flow-state where the goal is to find balance between player skills and game challenge. In general every video games provide difficulty settings. Difficulty settings that was provided usually in choice form such as easy, medium, and hard. However difficulty settings such as this usually make the game feels too flat, which make imbalance gameplay between player abilty to game challange. Dynamic Difficulty Adjustment (DDA) hopefully can solve the inconsistency gameplay from this problem. Dynamic Difficulty Adjustment is an alternative to a difficulty settings that needed to be set before playing a game, with DDA player do not need to set the difficulty manually before playing. DDA in this case will work as a substitute to the old difficulty settings system, because DDA work as a dynamic difficulty settings that will adjust automaticly to player ability. To support DDA system, Behaviour Tree will be used as method to make the algorithm. Testing is used by test play race between AI with DDA against static AI for 3 laps and distance difference will be recorded with every checkpoint passed. From this testing we know that Behaviour Tree will help DDA to make the game more fun and challenging. Behaviour Tree and DDA produce AI that have dyniamic difficulty so it can adapt to opponent abbilities.

Keywords: dynamic difficulty adjustment, DDA*, behaviour tree, racing game.*

DAFTAR ISI

BAB 1 PENDAHULUAN

1.1 Latar belakang

Video games merupakan hiburan dan tantangan. *Video games* tidak akan menyenangkan jika tidak ada tantangannya. Apabila tantangannya terlalu mudah maka akan jadi membosankan. Namun jika tantangannya terlalu sulit maka akan membuat kebanyakan orang frustasi. *Game* pada umumnya menyediakan Pengaturan tingkat kesulitan (*difficulty setting*) untuk mengatur seberapa menantang sebuah permainan. Tingkat kesulitan yang disediakan biasanya berkisaran dari mudah (*Easy*), sedang (*Medium*), sampai yang sulit (*Hard*). Adanya pengaturan tingkat kesulitan ini ditujukan untuk memilihkan tantangan yang cocok bagi pemula maupun yang sudah mahir (Kenneth, Torkil, & Long, 2011).

Tingkat kesulitan game yang statis bisa menyebabkan ketidaksetaraan antara pemain dengan tantangan dari game. Untuk menyelesaikan masalah ini dibuatlah sistem *Dynamic Difficulty Adjustment* (DDA) (Kenneth, Torkil, & Long, 2011). DDA adalah alternatif dari *Difficulty Settings* statis yang harus ditentukan oleh pemain sebelum memulai permainan, dengan adanya DDA pemain tidak perlu repot mengatur *difficulty* sebelum bermain. DDA berfungsi sebagai *Difficulty Settings* yang bekerja secara otomatis berdasarkan kemampuan pemain. Dengan adanya DDA pemain bisa merasakan permainan yang lebih menyenangkan dan menghibur. Hal ini berhubungan dengan *flow-state* yaitu keadaan ketika kemampuan pemain dan tantangan game setara (Kenneth, Torkil, & Long, 2011).

Pada penelitian sebelumnya yang dilakukan oleh (Mirna, Victor, & Luiz, 2016) "DDA ada sebagai alternatif dari tingkat kesulitan pada *game*. Yaitu mekanisme yang bisa mengatur perilaku AI berdasarkan kemampuan pemain." Setelah beberapa pengujian DDA yang diimplementasikan pada *game MOBA* didapati bahwa 85% dari total pengujian, mekanisme DDA berhasil menyamai kemampuan pemain dengan baik, sedangkan sisanya gagal. Dari 10% total pengujian gagal karena mekanisme DDA memakan banyak waktu untuk melakukan adaptasi dengan kemampuan pemain, sedangkan 5% sisanya gagal karena mekanisme DDA terlalu banyak melakukan adaptasi dalam waktu singkat sehingga menyababkan AI tidak berfungsi dengan baik (Mirna, Victor, & Luiz, 2016).

Salah satu metode yang bisa digunakan untuk DDA adalah metode *Behaviour Trees* (BT). Pada penelitian yang dilakukan oleh (Kenneth, Torkil, & Long, 2011) tentang DDA menggunakan *Behaviour Trees* pada *fighting game*, dapat disimpulkan bahwa *Behaviour Trees* menunjukkan hasil yang menjanjikan, AI dapat beradaptasi dengan beragam kemampuan pemain secara cepat dan bisa mengimbangi pemain hingga *game* berakhir. Pada penelitian yang dilakukan (Rabin, 2008) menyatakan "Alex J. Champandard merumuskan, pada umumnya BT dapat digambarkan sebagai struktur hirarki dari tugas (*task*). *Behaviour Trees* bisa digambarkan sebagai struktur yang menyimpan tugas/perintah untuk perilaku AI." Inilah yang nanti akan melakukan pengaturan *Difficulty* pada game.

Berdasarkan pada pernyataan di latar belakang, penelititan IMPLEMENTASI *DYNAMIC DIFFICULTY ADJUSTMENT* PADA *RACING GAME* MENGGUNAKAN METODE *BEHAVIOUR TREE* ini akan membuat AI yang dinamis dengan menggunakan sistem DDA. Jadi diharapkan pemain tidak akan merasa bosan dengan kemampuan lawan AI statis yang monoton, karena dengan menggunakan *AI* dinamis tingkat kesulitan akan beradaptasi seusai dengan kemampuan pemain.

1.2 Rumusan masalah

Rumusan masalah pada penelitian ini adalah :

1. Bagaimana membuat agen yang dapat mengubah tingkat kesulitan secara dinamis pada *racing game*?
2. Bagaimana perbandingan agen dengan tingkat kesulitan yang dinamis terhadap agen dengan tingkat kesulitan statis.

1.3 Tujuan

Tujuan pada peneilitian ini adalah :

1. Menerapkan agen yang dapat merubah tingkat kesulitan secara dinamis dengan sistem *Dynamic Difficulty Adjustment* menggunakan *Behaviour Tree* pada *Racing Game*.
2. Menguji tingkat perbandingan agen DDA dengan agen statis.

1.4 Manfaat

Manfaat dari penelitian ini untuk penulis adalah untuk menambah pengetahuan pengembangan AI DDA dalam *game*. Manfaat penelitian ini bagi masyarakat adalah untuk memberikan tantangan yang menarik dan tidak membosankan untuk para pemain *racing game*, tidak ada lagi permainan membosankan / lawan (AI) yang terlalu mudah dan tidak ada lagi permaian yang membuat frustasi / lawan (AI) yang terlalu sulit. Harapanya yaitu bagi para pemain baik dari yang pemula sampai yang sudah mahir pun bisa memiliki level *enjoyment* yang sama.

1.5 Batasan masalah

Batasan masalah :

1. Penelitian ini tidak membahas tentang pembuatan *racing game* karena menggunakan template *racing game* yang sudah ada.
2. Penelitian ini hanya akan membahas pembuatan perilaku AI pada *racing game*.
3. Penelitian ini hanya akan mengujikan kemampuan AI DDA ke AI statis dengan tingkat kesulitan (*Easy*, *Medium*, *Hard*).

1.6 Sistematika pembahasan

Untuk mencapai tujuan yang diharapkan, maka sistematika penulisan yang disusun dalam tugas metodologi penelitian ini adalah sebagai berikut:

BAB 1 Pendahuluan

Bab ini berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

BAB 2 Landasan Kepustakaan

Bab ini membahas teori-teori yang dibutuhkan dalam pemahaman permasalahan yang akan dibahas, serta teori-teori yang berhubungan dan diperlukan dalam penelitian ini.

BAB 3 Metodologi

Bab ini membahas tentang metode yang digunakan dalam penulisan yang terdiri dari studi literatur, wawancara, perancangan sistem, implementasi, pengujian dan analisis, dan pengambilan kesimpulan.

BAB 4 Perancangan

Bab ini membahas tentang perancangan AI untuk aplikasi *Racing Game*.

BAB 5 Implementasi

Bab ini membahas tentang penerapan dari rancangan yang dibuat pada bab sebelumnya dan analisis kebutuhan dari program.

BAB 6 Pengujian

Bab ini memuat tentang hasil pengujian dan analisis terhadap sistem yang telah dibuat.

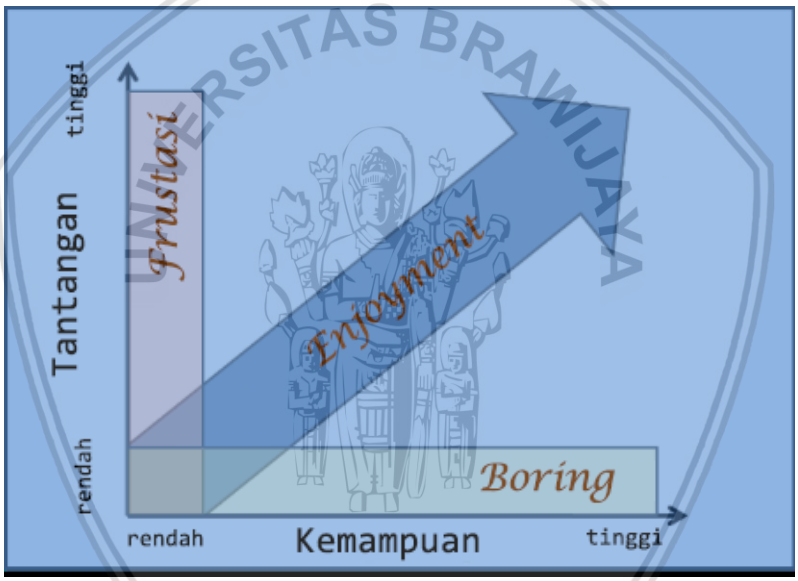
BAB 7 Penutup

Bab ini memuat tentang kesimpulan yang diperoleh dari pembuatan dan pengujian perangkat lunak yang dikembangkan dalam proposal skripsi ini serta saran – saran untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 *Dynamic Difficulty Adjustmen* (DDA)

Dynamic Difficulty Adjustment (DDA) adalah sistem yang mengatur tingkat kesulitan secara dinamis, hal ini berhubungan dengan *Enjoyment Flow,* Gambar 2.1, yang bertujuan untuk mendapatkan keseimbangan antara kemampuan pemain dengan tantangan *game*. Pada Gambar 2.1, dijelaskan bahwa tingkat *enjoyment* dapat dicapai apabila keseimbangan antara kemampuan dan tantangan adalah setara. Jika tingkat kemampuan lebih tinggi dari tingkat tantangan maka game akan menjadi membosankan, begitu juga sebaliknya jika tingkat kemampuan lebih rendah dari tingkat tantangan maka game akan membuat kebanyakan orang frustasi (Kenneth, Torkil, & Long, 2011). Dengan adanya DDA tingkat tantangan akan selalu berganti, beradaptasi dengan kemampuan pemain untuk mencapai *enjoyment* dalam permainan (Robin &



Gambar 2.1 *Enjoyment Flow* (Csikszentmihalyi**,** 1991**)**

Vernell, 2004).

Dalam industri *game*, DDA sudah diimplementasikan di beberapa tipe *game*. Pada *game First Person Shooter* (FPS) contohnya Half-Life 2, amunisi akan lebih sering *drop* dari musuh apabila pemain sedang dalam keadaan kekurangan amunisi, *first aid kit* jadi lebih mudah ditemukan jika *health point* pemain sedang dalam keadaan kritis (Robin & Vernell, 2004).

DDA pada *fighting games* berfungsi sebagai pengendali *skill* lawan (AI). AI akan menyusuaikan kemampuan sesuai dengan kemampuan pemain, jika pemain berkemampuan pemula (serangan banyak *miss*, tidak melakukan *block*, tidak mengerti *combo*, dsb) maka AI pun akan beradaptasi dengan kemampuan pemain (tidak melakukan *block* pada serangan pemain, menggunakan serangan kuat hanya sesekali, tidak mengeluarkan *special move*, tidak melakukan *combo expert*,

dsb). Begitu juga dengan pemain yang sudah mahir (mengerti bermacam macam *combo*, paham cara melakukan *block/parry*, bisa melakukan *special move*, dsb) maka AI juga akan beradaptasi dengan kemampuan pemain dan melakukan hal yang sama kepada pemain (Andrade, Santana, Ramalho, & Corruble, 2005).

Meskipun sistem DDA terlihat sangat bagus ketika di implementasikan dengan baik, meski begitu DDA tetap masih punya kekurangan. Semakin kompleks sebuah *game* maka akan semakin sulit untuk mengimplementasikan sistem DDA. Semakin sulit pula untuk mengatur perilaku AI sesuai keinginan. Dengan pengaturan tingkat kesulitan yang statis AI hanya akan berprilaku sesuai program, namun dengan implementasi DDA AI jadi bisa melakukan prilaku yang terkadang tidak diduga (Andrade, Santana, Ramalho, & Corruble, 2005).

2.2 AI Pada *Racing Game*

Racing game adalah *genre games* yang dimana pemain berlomba kecepatan melawan pemain lain atau melawan AI. Pada mulanya *racing game* hanya sekedar pemain satu dengan pemain lainya berlomba kecepatan mobil di sirkuit mencapai lap/garis finish. Seiring perkembangan jaman *racing game* sudah berkembang jauh, *racing game* saat ini rata-rata menyediakan *power up* dan bermacam-macam *skill* pada kendaraan, dan dengan permukaan sirkuit yang bermacam-macam seperti aspal, gurun, air, es, dsb (Simon & Nic, 2013).

Metode perilaku AI pada racing game tidak memiliki banyak macam, yang paling umum adalah *Finite-State Machine* (FSM), FSM ini sudah dianggap mencakupi kebutuhan perilaku *AI* pada *racing game*, FSM biasanya digunakan pada *classic racing game*. Adapula penggunan metode *Behaviour Tree* untuk perilaku *AI* pada *racing game* biasanya digunakan untuk perilaku *AI* yang sedikit lebih kompleks dibanding FSM, contohnya seperti pada Racing Game yang memiliki fitur *Power Up Crate*, *Nitro Pad*, *Obstacle Path*, dll (Simon & Nic, 2013).

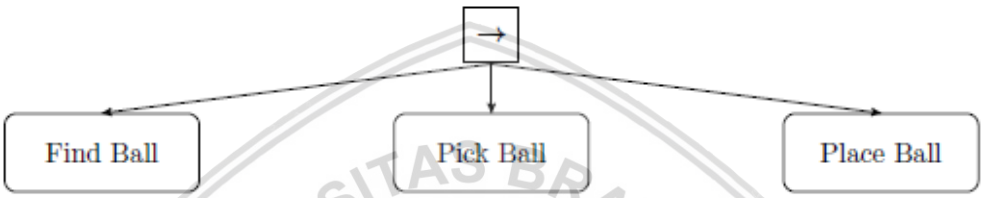
2.3 DDA Pada *Racing Game*

Dalam *racing game* biasanya tingkat kesulitan yang digunakan adalah tingkat kesulitan statis, jadi pemain harus menentukan terlebih dahulu tingkat kesulitan yang diinginkan sebelum memulai permainan. Dengan adanya DDA tidak perlu lagi adanya penggunaan pengaturan tingkat kesulitan.

Pada *racing game* implementasi DDA disini dibuat seperti karet gelang yang terhubung antara pemain dan lawan (AI). Pada sebuah *race* sering dijumpai kondisi dimana pemain dalam keadaan hampir *finish* dengan semua lawan (AI) tertinggal jauh, tiba - tiba lawan (AI) datang meluncur dengan kecepatan tinggi yang akhirnya merebut kemenangan dari pemain (Kenneth, Torkil, & Long, 2011). Implementasi DDA yang kurang baik dan kebanyakan pemain tidak menyukainya. Hal ini diibaratkan seperti karet gelang yang terhubung antara pemain dan lawan (AI), semakin tertinggal jauh AI maka akan semakin cepat lajunya untuk menutupi ketinggalan jarak antara AI dan pemain.

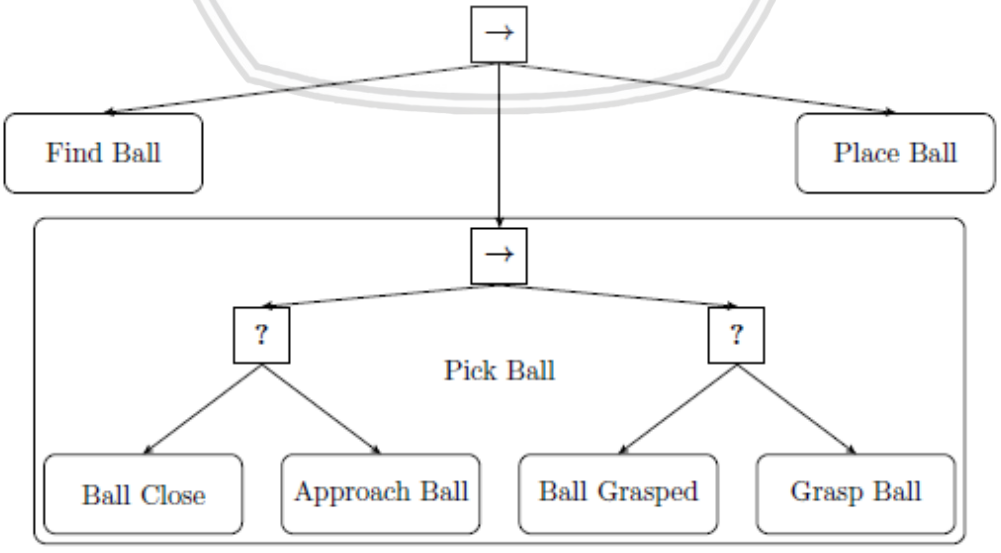
2.4 *Behaviour Tree* (BT)

Behaviour Tree adalah sebuah cara agen berpikir dan berperilaku menggunakan struktur hirarki yang menyimpan perilaku dan tingkah agen. BT dapat digambarkan sebagai pohon dari *node* hirarki yang mengatur alur perilaku yang dilakukan AI, seperti pada Gambar 2.2, *Behaviour Tree* yang menjalankan tugas secara berurutan dari *Find Ball*, *Pick Ball*, kemudian *Place Ball*. Pada pohon terdapat daun, daun disini dianalogikan sebagai perintah untuk AI, sedangkan rantingnya adalah berbagai macam node utilitas yang mengatur AI yang dipilih berdasarkan perintah dari daun, hingga turun ke batang pohon untuk memilih urutan perintah yang cocok pada situasi yang diberikan oleh daun.



Gambar 2.2 *Simple Tree* (Michele & Petter, 2017**)**

Behavior Tree bisa menjadi sangat kompleks, dengan *node* yang menjadi *sub-tree* yang dapat melakukan fungsi tertentu, hal ini membuat *developer* bisa membuat bermacam macam perilaku yang bisa di hubungkan satu sama lain yang menghasilkan perilaku AI yang sangat nyata. Perilaku AI bisa dkembangkan sampai sangat kompleks dengan adanya *sub-tree*. Dimulai dari perilaku-perilaku dasar, kemudian menyabang untuk mendapatkan metode alternatif untuk menyelesaikan perintah, jika satu cabang gagal dilakukan maka AI bisa kembali ke node sebelumnya dan memilih cabang lain hingga memenuhi kondisi yang diberikan seperti pada Gambar 2.3.



Gambar 2.3 *Simple Tree With Simple Branch* (Michele & Petter, 2017)

2.4.1 *Flow* Pada *Behaviour Trees*

Tiga fungsi inti yang dimiliki setiap *node* adalah *Success, Failure,* dan *Running*. *Success* adalah kondisi dimana perintah berhasil dieksuksi oleh *node*, sebaliknya *failure* adalah kondisi dimana perintah gagal dieksekusi oleh *node*, sedangkan *running* adalah kondisi dimana berhasil atau gagalnya perintah masih belum ditentukan (Alexander, Francisco, Matthew, Robert, & Norman, 2011), keterangan dapat diihat pada Tabel 2.1.

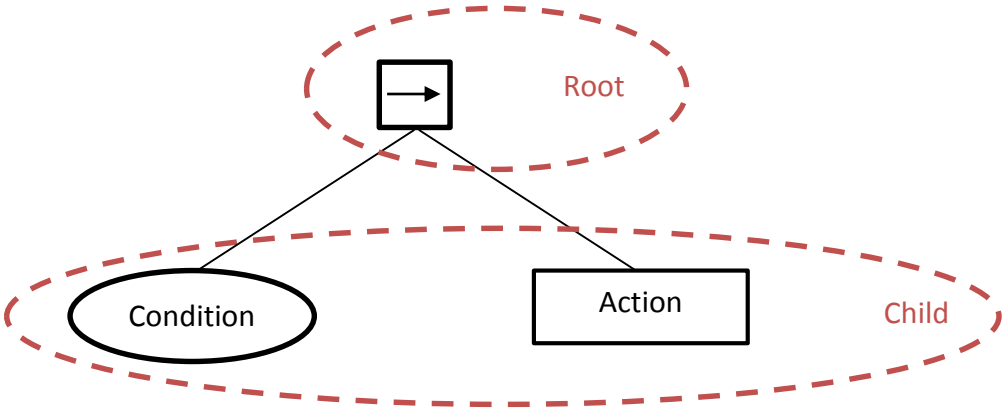
Tabel 2.1 *Flow Node Behaviour Tree* (Michele & Petter, 2017)

Tipe Node	Simbol	Sukses	Gagal	Berjalan
Fallback / Selector		Jika salah satu anak sukses	Jika semua anak gagal	Jika nilai kembalian satu anak berjalan
Sequence		Jika semua anak sukses	Jika salah satu anak gagal	Jika nilai kembalian satu anak berjalan
Parallel		Jika ≥ M anak sukses	Jika > N – M anak gagal	Selainnya
Action		Ketika selesai	Jika tidak bisa selesai	Saat berjalan
Condition		Jika kondisi terpenuhi	Jika kondisi tidak terpenuhi	-
Decorator		Sesuai keinginan	Sesuai keinginan	Sesuai keinginan

Fungsi dari node inilah yang menjadi kelebihan BT, fungsi ini yang memberikan informasi tentang apa yang selanjutnya harus dilakukan oleh *node*, contoh: *node* berjalan menjalankan perintah untuk berjalan ke tempat yang ditentukan, namun di tengah jalan ada *obstacle* (halangan) yang membuat *node* berjalan tidak bisa sampai ke tempat tujuan. Ketika hal ini terjadi fungsi *node* memberikan informasi *failure*, ketika *failure* terjadi maka *node* akan kembali ke posisi sebelumnya dan mencari jalan lain, ketika kondisi sudah dicapai maka *node* akan mengirimkan informasi *success* (Alexander, Francisco, Matthew, Robert, & Norman, 2011).

2.4.2 *Node* Pada *Behaviour Tree*

Behaviour Tree memiliki beberapa tipe *node* yang digunakan pada perancangan. pada Gambar 2.4 dapat dilihat *Behaviour Tree* sederhana, untuk menjalankan *Action* maka *Condition* harus dipenuhi, penjelasan tipe node dapat dililhat di Tabel 2.2.



Gambar 2.4 *Simple* BT

Tabel 2.2 Tipe Node Behaviour Tree

Tipe Node	**Penjelasan**
Composite	Adalah *node* yang mewakili satu cabang dari *Behaviour Tree* dan menjadi acuan untuk bagaimana cabang tersebut berjalan
Task	Adalah daun dari *Behaviour Tree*. *Node* yang menjalankan tugas dan tidak punya koneksi lanjutan
Decorator	Juga dikenal sebagai *Conditional.* Basanya ini terhubung dengan *node* lain dan menentukan apakan *node* itu bisa dijalankan atau tidak
Service	Biasanya terhubung dengan *node Composite* dan akan menjalankan tugas pada preferensi yang ditentukan selama cabang *tree* yang terhubung berhasil berjalan
Root	Adalah Starting point dari *Behaviour Tree*, *node* ini tidak bisa dihubungkan dengan *node Decorator* atau *node Service*.

2.5 NP Behave

NP Behave adalah *Behaviour Tree Library* yang bekerja berdasarkan kejadian yang ditentukan (*Event Driven Behaviour Tree*) untuk AI pada Unity. Berbeda dengan *Behaviour Tree* tradisional, *Event Driven Behaviour Tree* ini tidak perlu mondar-mandir ke *node root* setiap kali state berjalan, BT ini tidak akan berpindah state kecuali saat kondisi untuk state selanjutnya terpenuhi (Meniku, 2016). NP Behave dapat di unduh di: https://unitylist.com/p/3ob/NP-Behave

2.5.1 NP Behave *Blackboards*

Blackboards adalah kamus yang berisi kondisi yang bisa dilihat dan bertambah sesuai kondisi yang dijalankan AI, *Blackboards* berfungsi sebagai memori pada AI. Contoh *Event Driven Behaviour Tree* menggunakan fungsi *Blackboards* dapat dilihat pada Tabel 2.3.

Tabel 2.3 Contoh *Event Driven Behaviour Tree* (Meniku, 2016)

Event Driven Behaviour Tree

```
1   ...
2   behaviorTree = new Root(
3       new Service(0.5f, () =>
4   { behaviorTree.Blackboard["swap"]
5   = !behaviorTree.Blackboard.Get<bool>("swap"); },
6       new Selector(
7               new BlackboardCondition("swap",
8   Operator.IS_EQUAL, true, Stops.IMMEDIATE_RESTART,
9                   new Sequence(
10                      new Action(() => Debug.Log("foo")),
11                      new WaitUntilStopped()
12                  )
13              ),
14              new Sequence(
15                  new Action(() => Debug.Log("bar")),
16                  new WaitUntilStopped()
17              )
18          )
19      )
20  );
21  behaviorTree.Start();
22  ...
23
24
```

Penjelasan :

1. Pada baris 2 buat behaviour tree baru.
2. Pada baris 3 – 5 setiap 0,5 detik akan merubah nilai variabel *swap* pada *Blackboard* menjadi true dan false secara bergantian
3. Pada baris 6 buat *selector* baru.
4. Pada baris 7 – 8 cek kondisi variabel swap pada *Blackboard*, jika nilai variabel *swap* sama dengan *true* maka dilanjut ke *sequence* pada baris 9, jika nilai variabel swap sama dengan false maka dilanjut ke *sequence* pada baris 14.
5. Pada baris 9 buat *sequence* baru
6. Pada baris 10 *print* “foo”
7. Pada baris 14 buat *sequence* baru
8. Pada baris 15 *print* “bar”

2.5.2 Tipe *Node* NP Behave

NP Behave memiliki 4 tipe *node* yaitu *root node, composite node, decorator node, task node*. Penjelasan *node* pada NP Behave dapat dilihat pada tabel 2.4.

Tabel 2.4 Tipe *node* NP Behave

Node	Penjelasan
Root Node	Memiliki satu child, digunakan untuk memulai dan menghentikan seluruh Behaviour Tree.
Composite Node	Memiliki beberapa child, digunakan untuk memutuskan child mana yang akan dijalankan. Urutan dan hasil ditentukan oleh node ini.
Decorator Node	Memiliki satu child, digunakan untuk memodifikasi hasil dari child dan atau melakukan hal lain saat child berjalan.
Task Node	Daun dari Behaviour Tree, node yang menjalankan tugas.

2.6 Racing Game Starter Kit (RGSK)

Racing Game Starter Kit (RGSK) adalah *asset* pada unity *store* yang pada penelitian ini digunakan sebagai *template* untuk membuat *racing game*. Pada RGSK sudah termasuk *Simple AI*, *vehicle physics*, dan 7 tipe *racing* yaitu *Circuit, Lap Knockout, Time Trial, Speed Trap, Checkpoint, Elimination,* dan *Drift* (Intense Games, 2015). Racing Game Starter Kit dapat di unduh di: https://assetstore.unity.com/packages/templates/racing-game-starter-kit-22615

2.6.1 AI Pada RGSK

RGSK sudah menyediakan simple AI di dalam *asset*, ada 3 tingkat kesulitan yang disediakan yaitu *Easy, Medium,* dan *Hard*. Atribut yang digunakan oleh AI yaitu *Acceleration Sensitivity, Acceleration Wander, Brake Sensitivity,* dan *Nitro Probability* (Intense Games, 2015), penjelasan atribut AI dapat dilihat pada Tabel 2.5. *Template* AI ini akan digunakan sebagai acuan pada BAB 6.

Tabel 2.5 Atribut AI RGSK

Atribut	Penjelasan
Acceleration Sensitivity	Seberapa sensitif *AI* untuk melakukan *Accelerate*
Acceleration Wander	Seberapa cepat *AI* menaikkan *Accelerate* dari minimum *Accelerate* ke maksimum *Accelerate*
Brake Sensitivity	Seberapa sensitif *AI* untuk melakukan *Brake*
Nitro Probability	Probabilitas *AI* menggunakan *Nitro*

2.6.2 AI *Difficulty* pada RGSK

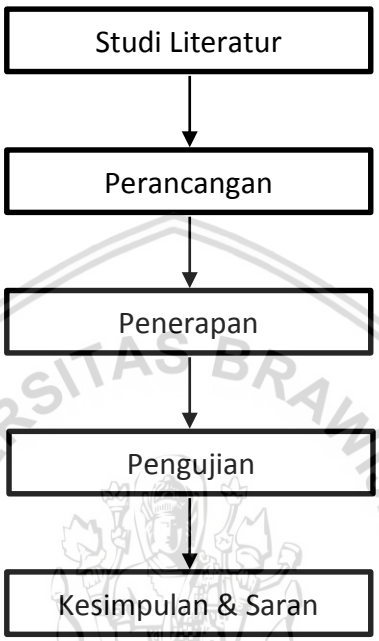
Ada 3 tingkat kesulitan AI pada RGSK yaitu *Easy, Medium,* dan *Hard* masing-masing tingkat kesulitan memiliki nilai atribut yang berbeda-beda. Atribut *Acceleration Sensitivity* memiliki nilai atribut dari 0 sampai 1, semakin besar nilainya maka akan semakin baik. Atribut *Acceleration Wander* memiliki nilai dari 1 sampai 0, semakin kecil nilainya maka akan semakin baik. Atribut *Brake Sensitivity* memiliki nilai dari 1 sampai 0, semakin besar nilainya maka semakin baik. Atribut *Nitro Probability* memiliki nilai dari 1 sampai 0, semakin besar nilainya maka akan semakin baik. Nilai atribut AI dapat dilihat di Tabel 2.6.

Tabel 2.6 Nilai Atribut AI RGSK

Atribut	**Tingkat Kesulitan**		
	Easy	***Medium***	***Hard***
Acceleration Sensitivity	0,5	0,8	1
Acceleration Wander	0,75 – 0,5	0,25 – 0,1	0
Brake Sensitivity	0,5	0,8	1
Nitro Probability	0 – 0,25	0,25 – 0,75	0,5 – 1

BAB 3 METODOLOGI

Dalam penelitian ini, metodologi penelitian yang harus dilakukan seperti studi literatur, analisis kebutuhan perangkat, perancangan, penerapan, pengujian, pengambilan kesimpulan seperti pada Gambar 3.1.



Gambar 3.1 Metodologi

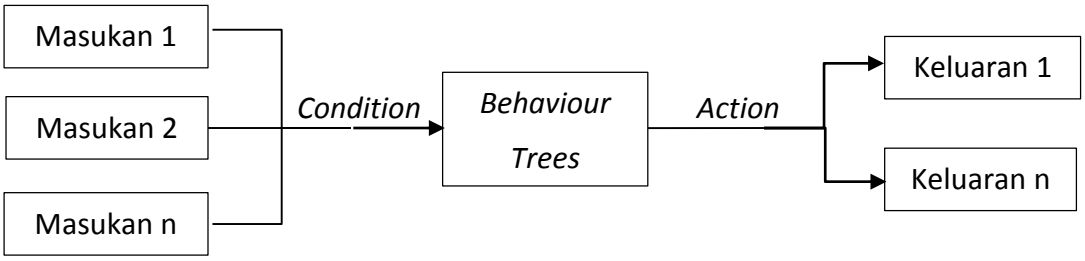
3.1 Studi Literatur

Pada tahapan ini, dilakukan kajian dari hal-hal yang bersangkutan dengan penelitian yang dilakukan. Studi literatur yang digunakan adalah sebagai berikut:

1. Teori *Dynamic Difficulty Adjustment*
2. *AI Dynamic Difficulty Adjustment*
3. Teori *Behaviour Tree*
4. NP Behave
5. Racing Game Starter Kit

3.2 Perancangan

Pada tahap ini, dilakukan peracangan terlebih dahulu agar tidak terjadi kesulitan dalam tahap penerapan.



Gambar 3.2 Perancangan

Dari Gambar 3.2 dapat dijelaskan bahwa, masukan merupakan variabel yang digunakan sebagai *node* kondisi pada *behaviour tree*, jika sebuah kondisi terpenuhi maka *action* yang bersangkutan akan dijalankan, yaitu keluaran yang berisi tugas/perilaku AI.

3.3 Implementasi

Proses yang dilakukan dalam tahap ini antara lain penerapan 5 tingkat kesulitan pada agen *AI*, penerapan metode *Behaviour Tree* pada agen *AI* DDA untuk menghasilkan agen *AI* yang dinamis.

3.4 Pengujian

Pengujian dilakukan dengan uji coba *AI* DDA pada *AI* statis dengan tingkat kesulitan *Easy, Medium,* dan *Hard.* Untuk membuktikan apakah *AI* DDA sudah berjalan dengan baik, yaitu apakah tingkat kesulitan bisa beradaptasi sesuai dengan kemampuan *AI* statis yang diujikan.

3.5 Kesimpulan

Pada tahap terakhir dilakukan pengambilan kesimpulan berdasarkan hasil penelitian pada tahap perancangan, penerapan, dan pengujian sistem. Yaitu tingkat efektifitas penggunaan *AI* DDA dengan metode *Behaviour Tree* pada *racing game*, dan hasil uji coba pada *AI* statis dengan *difficulty Easy, Medium,* dan *Hard*.

BAB 4 PERANCANGAN

4.1 Perancangan Agen AI

Sebagaimna yang sudah dijelaskan sebelumnya pada bab 2, genre permainan yang digunakan dalam penelitian ini adalah Racing Game. Agen AI yang dimaksud disini adalah karakter yang akan menjadi lawan pemain dalam balapan. AI disini memiliki 5 tingkat kesulitan, dimana setiap tingkat kesulitan akan memiliki *attribute value* yang berbeda-beda. Tingkat kesulitan AI akan ditentukan dari kempampuan pemain dan akan berganti-ganti sesuai tingkat kemampuan pemain. Penjelasan mengenai Atrribute AI dapat dilihat di Tabel 4.1.

Tabel 4.1 Nilai Atribut AI DDA

Atribut	**Tingkat Kesulitan**				
	Dif1	**Dif2**	**Dif3**	**Dif4**	**Dif5**
Acceleration Sensitivity	0,2	0,4	0,6	0,8	1
Acceleration Wander	0,6 – 0,5	0,5 – 0.4	0,4 – 0,3	0,3 – 0,2	0,2 – 0,1
Brake Sensitivity	0,2	0,4	0,6	0,8	1
Nitro Probability	0 – 0,1	0,1 – 0,2	0,2 – 0,4	0,4 – 0,8	0,5 – 1

4.2 Penentuan Tingkat Kesulitan AI

Untuk menentukan tingkat kesulitan AI, dibutuhkan masukan dari pemain. Masukan dari pemain disini berupa *value* dari poin yang dilakukan pemain di dalam game. Penjelasan dapat dilihat di Tabel 4.2, 4.3.

Tingkat kesulitan AI akan diproses ulang setiap checkpoint dilewati. jadi tingkat kesulitan AI akan terus berganti-ganti secara dinamis sesuai dengan kemampuan pemain.

Tabel 4.2 Player Input

Masukan	**Penjelasan**
Checkpoint	Adalah *trigger* pengukur jarak antara AI dan pemain.
DIstance	Adalah jarak antara kendaraan AI dan Pemain dalam unit jarak pada unity.

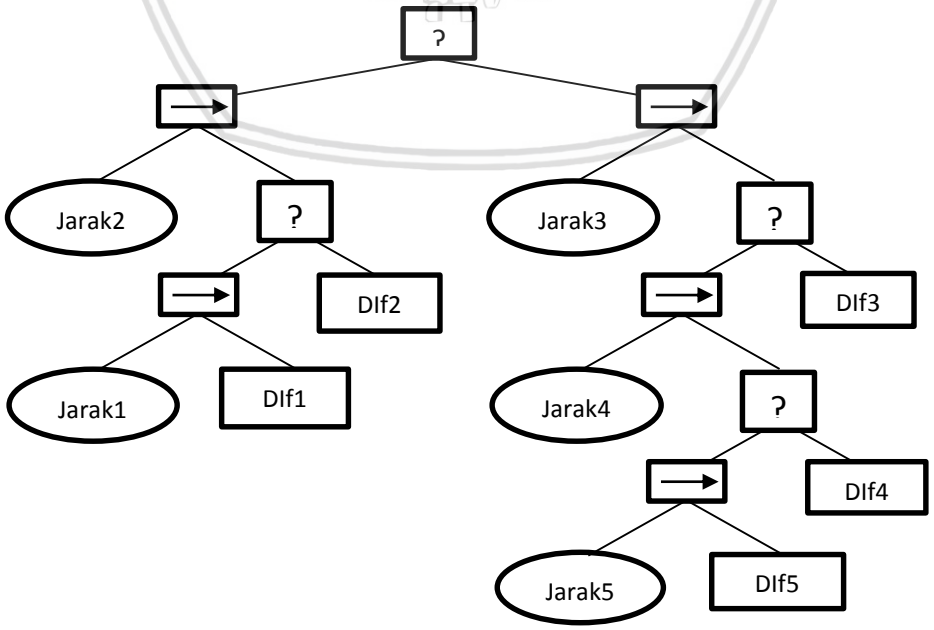
Tabel 4.3 Penghitungan Jarak

Jarak	Tingkat Kesulitan	Penjelasan
Jarak1 < (-50)	Dif1	Jarak antara kendaraan Pemain dan AI lebih kecil dari (-50) unit jarak pada unity
Jarak2 < (0)	Dif2	Jarak antara kendaraan Pemain dan AI lebih kecil dari (0) unit jarak pada unity
Jarak3 > 50	Dif3	Jarak antara kendaraan Pemain dan AI lebih besar dari 50 unit jarak pada unity
Jarak4 > 100	Dif4	Jarak antara kendaraan Pemain dan AI lebih besar dari 100 unit jarak pada unity
Jarak5 > 150	Dif5	Jarak antara kendaraan Pemain dan AI lebih besar dari 150 unit jarak pada unity

Pada Tabel 4.3 dapat dilihat yaitu jika jarak antara pemain dan AI DDA lebih kecil dari (-50), maka tingkat kesulitan AI DDA akan diatur menjadi Dif1, jika jarak antara pemain dan AI DDA lebih kecil dari (0) dan tidak lebih kecil dari (-50) maka tingkat kesulitan akan diatur menjadi DIf2, dan begitu seterusnya.

4.3 Perancangan Algoritma DDA Behaviour Tree

Rancangan *Behaviour Tree* untuk algoritma DDA dapat dilihat pada Gambar 4.1.



Gambar 4.1 *Behaviour Tree* Algoritma DDA

Cara membaca *Behaviour Tree* adalah dibaca dari atas ke bawah, dari kiri ke kanan. Seperti pada Gambar 4.1 dapat dilihat setelah *node selector pertama (root)* masuk ke *node sequence* sebelah kiri, kemudian cek kondisi Jarak2 apakah terpenuhi, jika terpenuhi maka lanjut ke *node sequence*, kemudian cek kondisi Jarak1, jika terpenuhi maka jalankan fungsi Dif1, jika kondisi Jarak1 tidak terpenuhi, maka *return* nilai ke *node selector* kemudian jalankan fungsi Dif2. Jika pada cek kondisi Jarak2 tidak terpenuhi, maka return nilai ke *node selector* awal kemudian lanjut ke *node sequence* sebelah kanan. Pada *node sequence* sebelah kanan lanjut ke cek kondisi Jarak3, jika terpenuhi maka lanjut ke *node selector* dan lanjut ke *node sequence*. Kemudian cek kondisi Jarak4, jika tidak terpenuhi maka jalankan fungsi Dif3. Jika cek kondisi Jarak4 terpenuhi maka lanjut ke *node selector* dan lanjut ke *node sequence*, kemudian cek kondisi Jarak5, jika terpenuhi maka jalankan fungsi Dif5. Jika cek kondisi Jarak5 tidak terpenuhi maka jalankan fungsi Dif4.

4.4 Perancangan Pengujian

Pada penelitian ini akan dilakukan pengujian efektifitas antara AI DDA terhadap AI statis dan pengujian antara AI DDA dan AI statis melawan pemain. Dari setiap skenario akan dilakukan uji coba balap sebanyak 3 *lap* dengan 9 checkpoint dari setiap *lap* dan dicatat nilai jarak tempuh dari setiap checkpoint yang dilewati. Nilai jarak dari masing-masing difficulty akan dijumlahkan dan dihitung rata – rata dari setiap skenario, kemudian dibuat grafik dari setiap skenario.

4.4.1 Pengujian AI DDA Melawan AI Statis

Pada pengujian AI DDA melawan AI statis akan dilakukan dengan 3 skenario, tiap skenario AI DDA akan melawan AI dengan tingkat kesulitan berbeda-beda, yaitu:

1. AI DDA melawan AI dengan tingkat kesulitan *Easy.*
2. AI DDA melawan AI dengan tingkat kesulitan *Medium.*
3. AI DDA melawan AI dengan tingkat kesulitan *Hard.*

4.4.2 Pengujian AI DDA dan AI Statis Melawan Pemain

Pada pengujian AI DDA dan AI statis melawan pemain akan dilakukan dengan 4 skenario, tiap skenario pemain akan melawan AI dengan tingkat kesulitan berbeda-beda, yaitu:

1. Pemain melawan AI DDA
2. Pemain melawan AI dengan tingkat kesulitan *Easy.*
3. Pemain melawan AI dengan tingkat kesulitan Medium.
4. Pemain melawan AI dengan tingkat kesulitan *Hard.*

BAB 5 IMPLEMENTASI

5.1 Penentuan Spesifikasi Sistem

Pada bab ini dijelaskan tentang kebutuhan sistem apa saja yang berguna untuk penerapan dan pengujian, yaitu ada pada tabel 5.1, 5.2.

Tabel 5.1 Spesifikasi Perangkat Lunak

Operating System	Windows 10 Pro
Program Language	C#
Software Development Kit	Unity 2017.2.1f1 Personal (64bit)
Software Editor	Visual Studio / Monodevelop

Tabel 5.2 Spesifikasi Perangkat Keras

Processor	Intel® Core™ i3-3217U @1.80GHz
Memory (RAM)	4096MB
Graphic Card	Intel® HD Graphics 4000

5.2 Penerapan Agen AI

Agen AI disini menggunakan *Game Object* pada *game engine* Unity. Terdapat dua Agen AI yaitu AI Statis yang menggunakan difficulty statis, dan AI DDA yaitu AI yang sudah diterapkan sistem Dynamic Difficulty Adjustment. Dapat dilihat pada Gambar 5.1, AI statis menggunkan mobil berwarna biru dan AI DDA menggunakan mobil berwarna putih.

Gambar 5.1 AI DDA melawan AI Statis

5.3 Penerapan Hitung Jarak

Jarak dihitung antara progres vehicle player dikurangi progres vehicle AI. Kode penerapan hitung jarak dapat dilihat pada Tabel 5.3.

Tabel 5.3 Kode Hitung Jarak

Hitung Jarak

```
1   private void UpdatePlayerDistance()
2           {
3               jarak = GetDistanceFromPlayer();
4               jarak =
5   opponent.GetComponent<ProgressTracker>().progressDistance -
6   progressTracker.progressDistance;
7               if ( tag == "Player")
8               {
9                   behaviourTree.Blackboard["playerDistance"] =
10  jarak;
11                  Debug.Log("Jarak: " + jarak);
12                  Debug.Log("Difficulty: " + difficulty);
13              }
14          }
```

Penjelasan:

1. Pada baris 4 – 6 menghitung selisih jarak antara AI DDA dengan pemain berdasarkan total progres.
2. Pada baris 9 – 12 menyimpan nilai jarak ke *blackboard*, dan menampilkan nilai jarak serta tingkat kesulitan AI DDA ke *console*

5.4 Penerapan Checkpoint Trigger

Pemanggilan DDA dilakukan setiap kali vehicle melewati checkpoint. Kode untuk Checkpoint trigger dapat dilihat pada Tabel 5.4.

Tabel 5.4 Kode Checkpoint Trigger

Checkpoint Trigger

```
1   void OnTriggerEnter(Collider other)
2         {
3               //Time Stamp
4               if (other.tag == "TimeStamp")
5               {
6                 UpdatePlayerDistance();
7               }
8         }
```

Penjelasan:

1. Pada baris 4 akan menjalankan tugas ketika objek dengan tag “TimeStamp” di trigger. TimeStamp adalah tag untuk objek checkpoint.
2. Pada baris 3 melakukan UpdatePlayerDIstance.

5.5 Difficulty AI DDA

Pada difficulty DDA ini dibuat 5 difficulty AI dari paling mudah ke paling sulit. Kode untuk Difficulty dapat dilihat pada Tabel 5.5.

Tabel 5.5 Kode Difficulty AI DDA

Difficulty AI DDA

```
1   public void SetDifficulty(AiDifficulty _difficulty)
2      {
3         difficulty = _difficulty;
4         switch (_difficulty)
5         {
6          case AiDifficulty.dif1:
7                accelerationSensitivity = 0.2f;
8                brakeSensitivity = 0.2f;
9                accelerationWander = Random.Range(0.5f, 0.6f);
10               nitroProbability = Random.Range(0f, 0.1);
11               break;
12
13        case AiDifficulty.dif2:
14               accelerationSensitivity = 0.4f;
15               brakeSensitivity = 0.4f;
16               accelerationWander = Random.Range(0.4f, 0.5f);
17               nitroProbability = Random.Range(0.1f, 0.2f);
18               break;
19
20        case AiDifficulty.dif3:
21               accelerationSensitivity = 0.6f;
22               brakeSensitivity = 0.6f;
23               accelerationWander = Random.Range(0.3f, 0.4f);
24               nitroProbability = Random.Range(0.2f, 0.4f);
25               break;
26
27        case AiDifficulty.dif4:
28               accelerationSensitivity = 0.8f;
29               brakeSensitivity = 0.8f;
30               accelerationWander = Random.Range(0.2f, 0.3f);
31               nitroProbability = Random.Range(0.4f, 0.8f);
32               break;
33
34        case AiDifficulty.dif5:
35               accelerationSensitivity = 1f;
36               brakeSensitivity = 1f;
37               accelerationWander = Random.Range(0.1f, 0.2f);
38               nitroProbability = Random.Range(0.5f, 1f);
39               break;
40        }
41     }
```

Penjelasan:

1. Pada baris 4 fungsi *switch case* adalah untuk memeriksa nilai variabel _difficulty untuk menentukan arah percabangan
2. Baris 6 – 11 berfungsi untuk mengatur nilai atribut AI DDA pada dif1.
3. Baris 13 – 18 berfungsi untuk mengatur nilai atribut AI DDA pada dif2.
4. Baris 20 – 25 berfungsi untuk mengatur nilai atribut AI DDA pada dif3.
5. Baris 27 – 32 berfungsi untuk mengatur nilai atribut AI DDA pada dif4.
6. Baris 34 – 39 berfungsi untuk mengatur nilai atribut AI DDA pada dif5.

5.6 Penerapan Algoritma DDA Menggunakan Behaiour Tree

Merupakan representasi *Behaviour Tree* pada Gambar 4.1 dalam bentuk kode. Kode untuk penerapan Algoritma DDA menggunakan metode Behaiour Tree dapat dilihat pada Tabel 5.6.

Tabel 5.6 Kode Behaviour DDA

Behaviour Tree DDA

```
private Root CreateBehaviourTree()
        {
        return new Root(
                new Selector(
                 new Sequence(
                  new BlackboardCondition ("playerDistance",
Operator.IS_SMALLER, 0f, Stops.IMMEDIATE_RESTART,
                   new Selector(
                    new Sequence(
                     new BlackboardCondition
("playerDistance", Operator.IS_SMALLER, -50f,
Stops.IMMEDIATE_RESTART,
                      new Action(() =>
SetDifficulty(AiDifficulty.dif1))
                     )
                    ),
                   new Action(() =>
SetDifficulty(AiDifficulty.dif2)),
                   new WaitUntilStopped()
                    )
                   )
                  ),
                 new Sequence(
                  new BlackboardCondition("playerDistance",
Operator.IS_GREATER, 50f, Stops.IMMEDIATE_RESTART,
                   new Selector(
                    new Sequence(
                     new BlackboardCondition("playerDistance",
Operator.IS_GREATER, 100f, Stops.IMMEDIATE_RESTART,
                      new Selector(
                       new Sequence(
                        new BlackboardCondition
("playerDistance", Operator.IS_GREATER, 150f,
Stops.IMMEDIATE_RESTART,
                         new Action(() =>
SetDifficulty(AiDifficulty.dif5))
                        )
                       ),
                      new Action(() =>
SetDifficulty(AiDifficulty.dif4)),
                      new WaitUntilStopped()
                       )
                      )
                     ),
                   new Action(() =>
SetDifficulty(AiDifficulty.dif3)),
                   new WaitUntilStopped()
                   )
                 )
             )
          )
```

```
51          );
52  }
```

Penjelasan:

1. Pada baris 3 inisialisasi *root Behaviour Tree.*
2. Pada baris 4 inisialisasi *node selector* pertama.
3. Pada baris 5 inisialisasi *node sequence.*
4. Pada baris 6 – 7 inisisalisasi `BlackboardCondition` memeriksa nilai variabel `playerDistance` apabila nilai lebih kecil dari 0 maka *Behaviour Tree* menjalankan *node* berikutnya.
5. Pada baris 8 inisialisasi *node selector.*
6. Pada baris 9 inisialisasi *node sequence.*
7. Pada baris 10 – 12 inisisalisasi `BlackboardCondition` memeriksa nilai variabel `playerDistance` apabila nilai lebih kecil dari -50 maka *Behaviour Tree* menjalankan *node* berikutnya.
8. Pada baris 13 – 14 mengatur tingkat kesulitan menjadi dif2.
9. Pada baris 17 – 19 mengatur tingkat kesulitan menjadi dif1.
10. Pada baris 23 inisialisasi *node sequence*.
11. Pada baris 24 – 25 inisisalisasi `BlackboardCondition` memeriksa nilai variabel `playerDistance` apabila nilai lebih besar dari 50 maka *Behaviour Tree* menjalankan *node* berikutnya.
12. Pada baris 26 inisialisasi *node selector.*
13. Pada baris 27 inisialisasi *node sequence.*
14. Pada baris 28 – 29 inisisalisasi `BlackboardCondition` memeriksa nilai variabel `playerDistance` apabila nilai lebih besar dari 100 maka *Behaviour Tree* menjalankan *node* berikutnya.
15. Pada baris 30 inisialisasi *node selector.*
16. Pada baris 31 inisialisasi *node sequence.*
17. Pada baris 32 – 34 inisisalisasi `BlackboardCondition` memeriksa nilai variabel `playerDistance` apabila nilai lebih besar dari 150 maka *Behaviour Tree* menjalankan *node* berikutnya.
18. Pada baris 35 – 36 mengatur tingkat kesulitan menjadi dif5.
19. Pada baris 39 – 41 mengatur tingkat kesulitan menjadi dif4.
20. Pada baris 45 – 47 mengatur tingkat kesulitan menjadi dif3.

5.7 Inisialisasi Game

Kode pada inisialisai *game* berisi fungsi-fungsi yang dijalankan ketika game dimulai. Kode inisialisasi mulai game bisa dilihat pada Tabel 5.7.

Tabel 5.7 Kode Inisialisasi Game

Fungsi yang diinisiasi saat *game* dijalankan

```
1   void Start()
2     {
3         SetDifficulty(difficulty);
4         player = GameObject.FindGameObjectWithTag("Player");
5         opponent = GameObject.FindGameObjectWithTag("Opponent");
6         behaviourTree = CreateBehaviourTree();
```

```
7           blackboard = behaviourTree.Blackboard;
8                 if (tag == "Player")
9               {
10                  randomBehaviour = false;
11                  behaviourTree.Start();
12              }
13              RandomizeAIBehaviour();
14  }
```

Penjelasan:

1. Baris 3 berfungsi untuk mengatur tingkat kesulitan di awal mulai.
2. Baris 4 berfungsi untuk menjadikan variabel player sebagai `GameObject` dengan tag `"Player"`.
3. Baris 5 berfungsi untuk menjadikan variabel player sebagai `GameObject` dengan tag `"Player"`.
4. Pada baris 6 membuat *Behaviour Tree*.
5. Pada baris 7 inisialisasi *blackboard*.
6. Pada baris 8 – 12 mematikan fungsi `randomBehaviour` serta mulai menjalankan *Behaviour Tree* untuk `GameObject` dengan tag `"Player"`.
7. Pada baris 13 memanggil fungsi `RandomizeAIBehaviour`.

BAB 6 PENGUJIAN DAN ANALISIS

6.1 Pengujian DDA

Pada pengujian DDA, akan dilakukan pengujian yang bertujuan untuk mengevaluasi apakah hasil dari DDA menggunakan *Behaviour Tree* yang sudah diimplementasikan dapat memberikan hasil yang sesuai dengan tujuan penelitian, yaitu menguji tingkat perbandingan antara AI DDA melawan AI statis dan AI DDA melawan pemain.

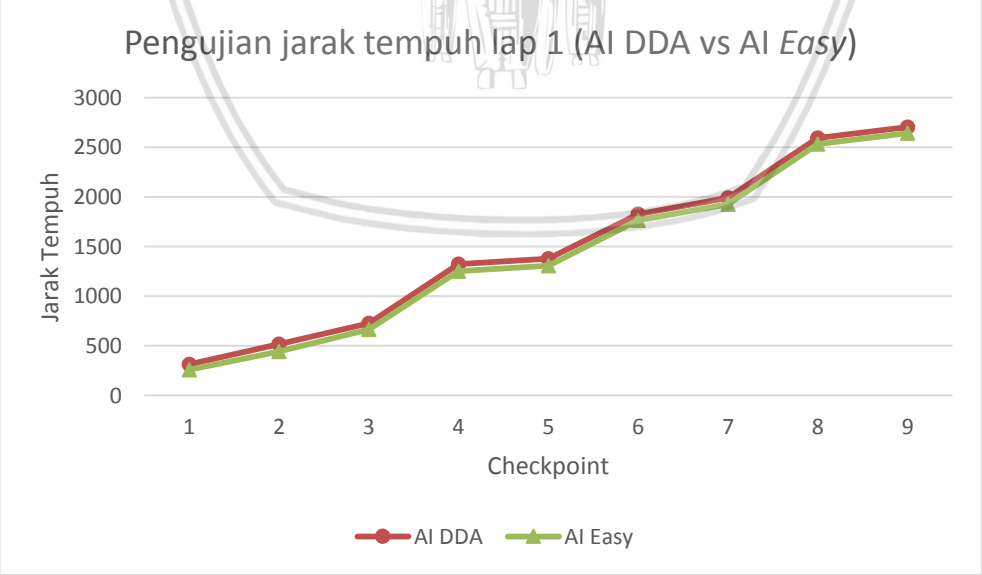
6.2 Pengujian AI DDA Melawan AI Statis

Pengujian akan dilakukan dengan cara membandingkan selisih jarak antara AI dengan tingkat kesulitan yang dinamis (AI DDA) melawan AI dengan tingkat kesulitan statis (AI statis). Ada 3 skenario pengujian yang akan dilakukan yaitu AI DDA melawan AI statis dengan tingkat kesulitan *Easy*, AI DDA melawan AI statis dengan tingkat kesulitan *Medium*, dan AI DDA melawan AI statis dengan tingkat kesulitan *Hard*.

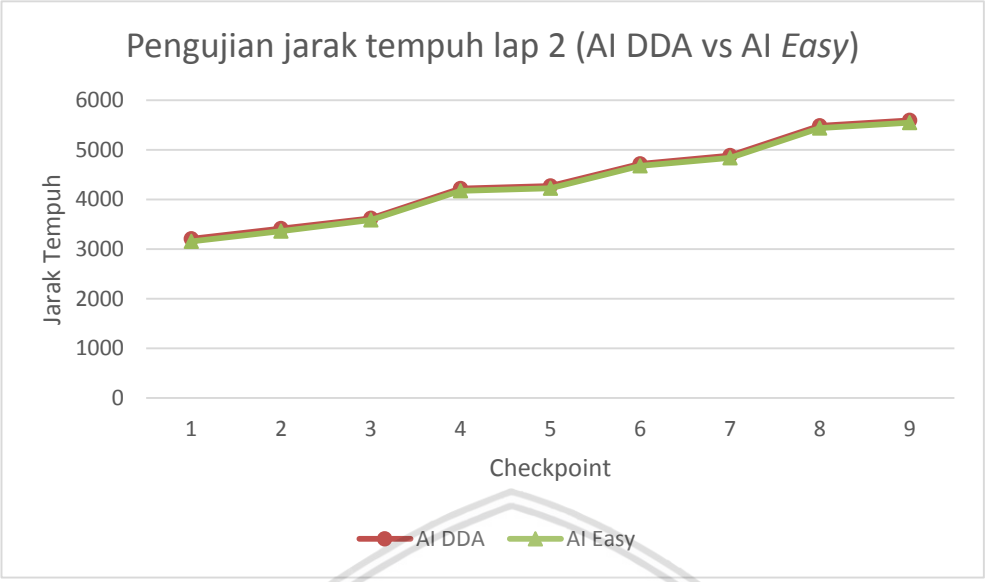
Untuk setiap skenario akan diujikan sebanyak 3 *lap* dengan 9 *checkpoint* di setiap lap, akan dilakukan perbandingan jarak tempuh antara AI DDA dengan AI statis. Satuan jarak yang digunakan adalah satuan jarak pada unity.

6.2.1 Pengujian Jarak Tempuh (AI DDA vs AI *Easy*)

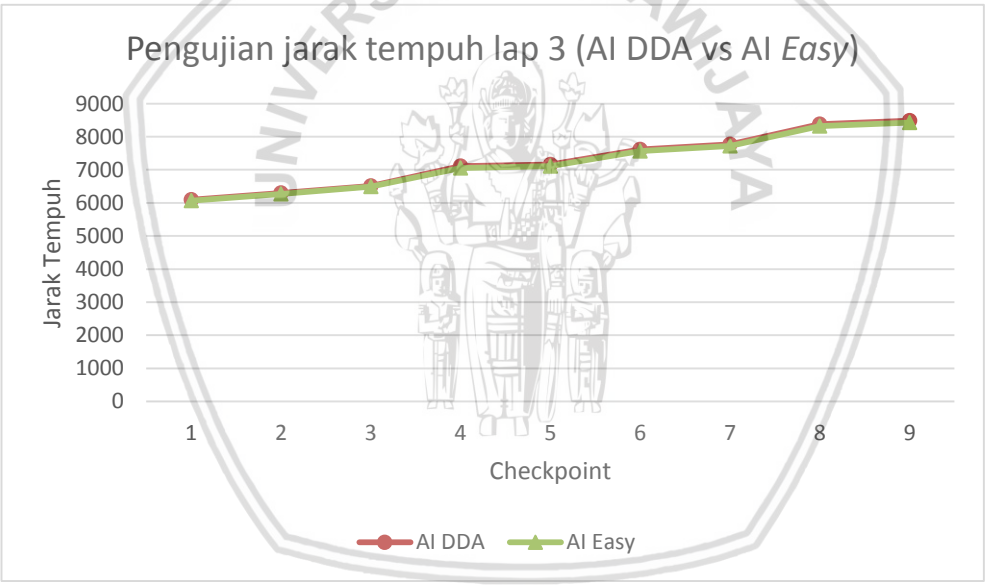
Hasil pengujian jarak tempuh untuk AI DDA melawan AI statis dengan tingkat kesulitan *Easy* dapat dilihat pada Gambar 6.1, Gambar 6.2, dan Gambar 6.3.



Gambar 6.1 Grafik pengujian jarak tempuh lap 1 (AI DDA vs AI *Easy*)



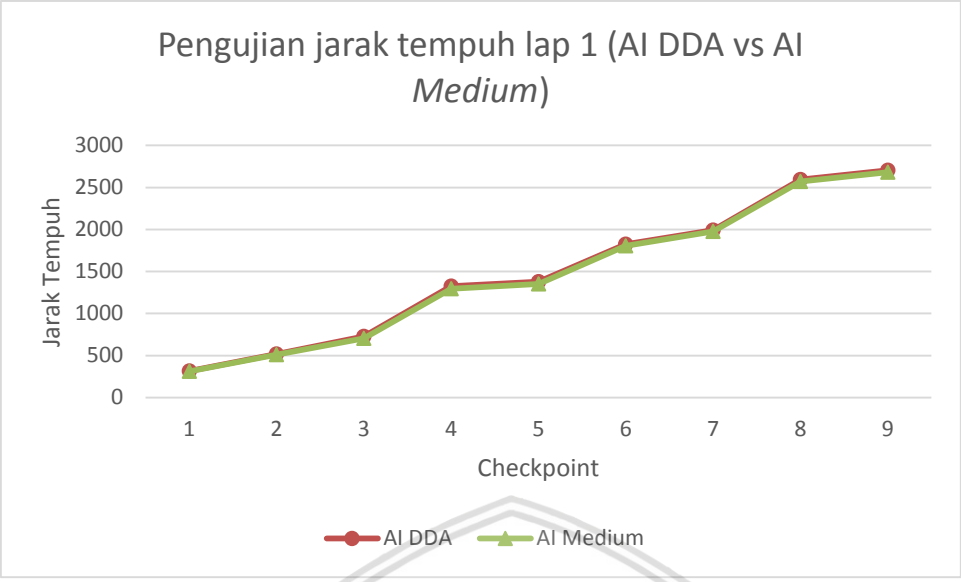
Gambar 6.3 Grafik pengujian jarak tempuh lap 2 (AI DDA vs AI *Easy*)



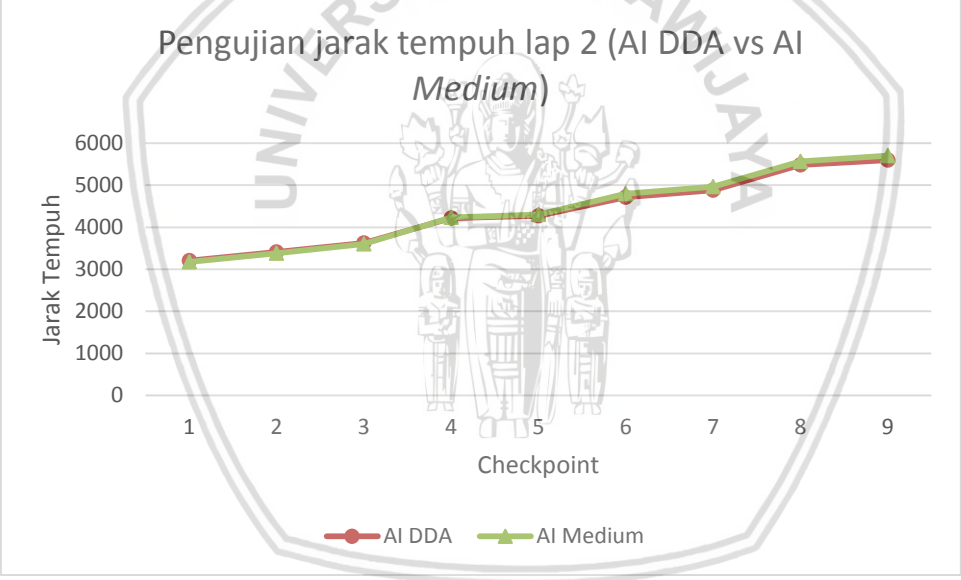
Gambar 6.2 Grafik pengujian jarak tempuh lap 3 (AI DDA vs AI *Easy*)

6.2.2 Pengujian Jarak Tempuh (AI DDA vs AI *Medium*)

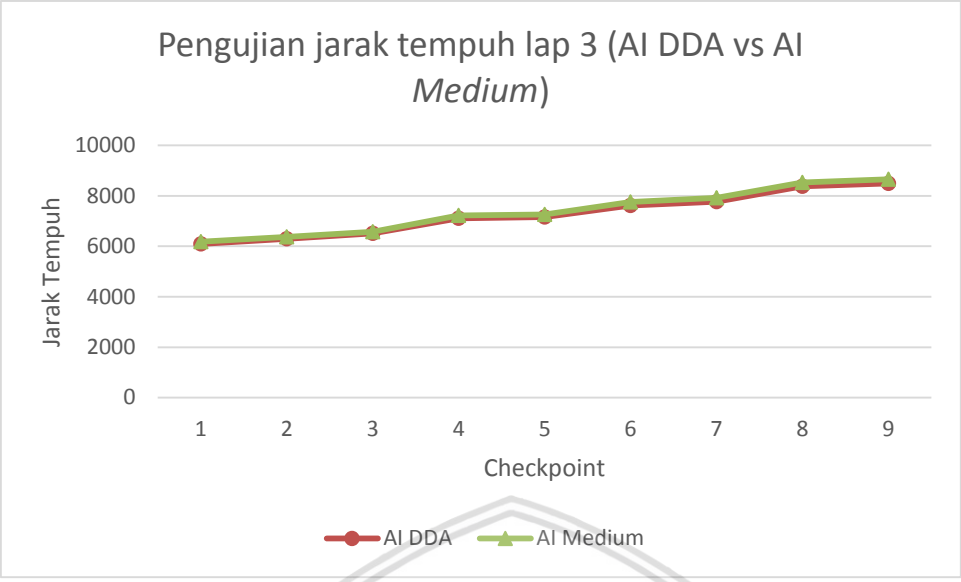
Hasil pengujian jarak tempuh untuk AI DDA melawan AI statis dengan tingkat kesulitan *Medium* dapat dilihat pada Gambar 6.4, Gambar 6.5, dan Gambar 6.6.



Gambar 6.4 Grafik pengujian jarak tempuh lap 1 (AI DDA vs AI *Medium*)



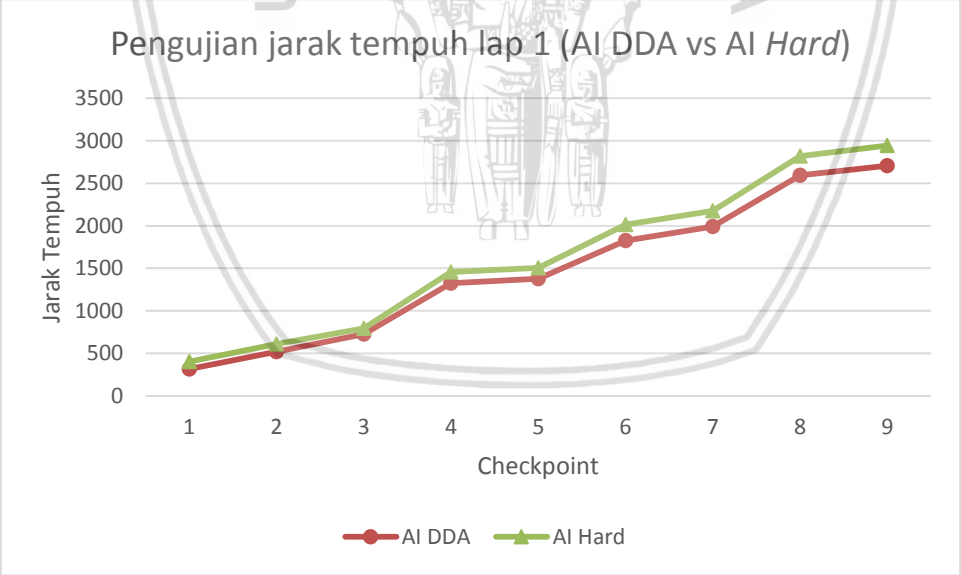
Gambar 6.5 Grafik pengujian jarak tempuh lap 2 (AI DDA vs AI *Medium*)



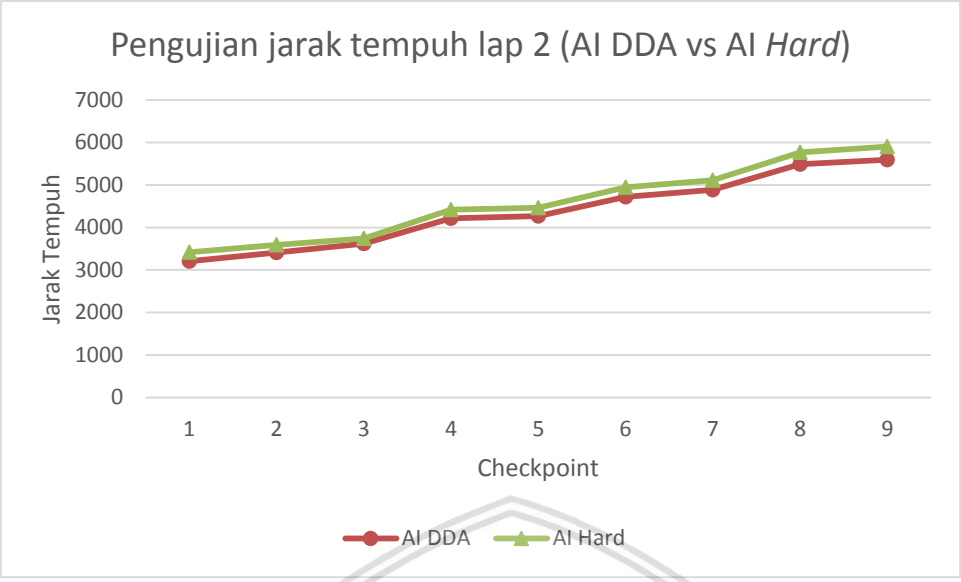
Gambar 6.6 Grafik pengujian jarak tempuh lap 3 (AI DDA vs AI *Medium*)

6.2.3 Pengujian Jarak Tempuh (AI DDA vs AI *Hard*)

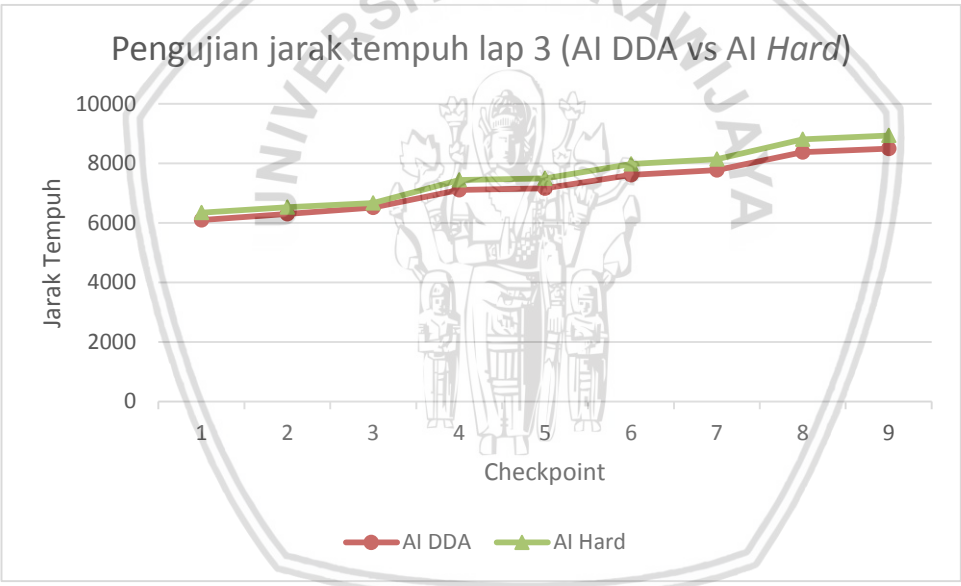
Hasil pengujian jarak tempuh untuk AI DDA melawan AI statis dengan tingkat kesulitan *Hard* dapat dilihat pada Gambar 6.7, Gambar 6.8, dan Gambar 6.9.



Gambar 6.7 Grafik pengujian jarak tempuh lap 1 (AI DDA vs AI *Hard*)



Gambar 6.9 Grafik pengujian jarak tempuh lap 2 (AI DDA vs AI *Hard*)



Gambar 6.8 Grafik pengujian jarak tempuh lap 3 (AI DDA vs AI *Hard*)

6.3 Pengujian AI DDA dan AI Statis Melawan Pemain

Pengujian dilakukan dengan cara mengujikan *game racing* yang sudah menggunakan AI DDA dengan metode *behaviour tree* dan *game racing* yang menggunakan AI statis kepada penguji secara langsung. Penguji adalah pemain yang berpengalaman dengan kemampuan yang baik. Berikut merupakan

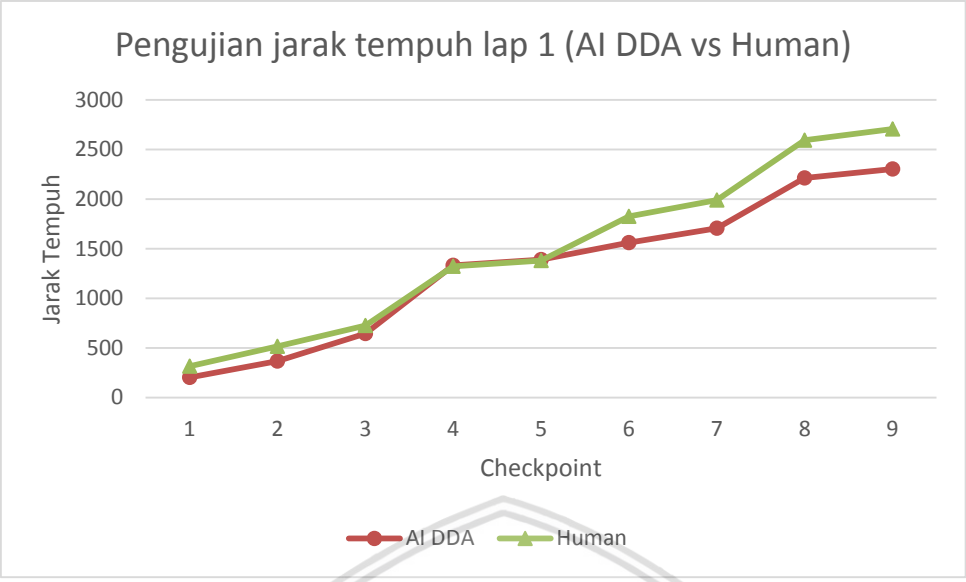
beberapa racing game yang pernah dimainkan penguji sebagai bukti untuk membantu menyatakan kemampuan pemain, dapar dilihat pada Tabel 6.1.

Tabel 6.1 *Game Racing* yang pernah dimainkan

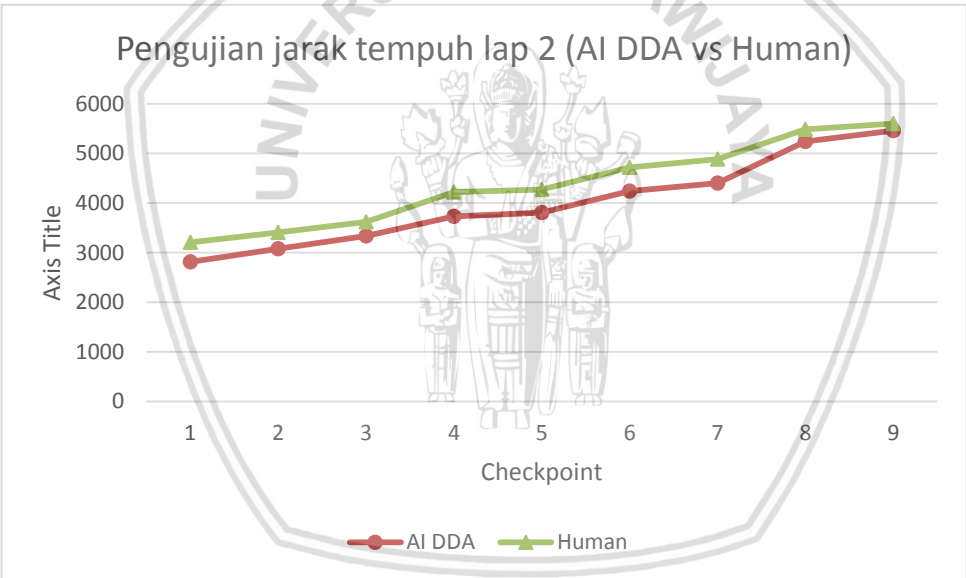
No.	Judul *Game*	Status
1	Nascar Rumble	Belum Tamat
2	Crash Team Racing	Tamat
3	Crash Tag Team Racing	Tamat
4	Chocobo Racing	Tamat
5	Bomberman Fantasy Race	Tamat
6	Need For Speed Most Wanted	Tamat
7	Need For Speed Underground	Tamat
8	Need For Speed Underground 2	Belum Tamat
9	Need For Speed Undercover	Tamat
10	Need For Speed Carbon	Tamat
11	Need For Speen Pro Street	Tamat
12	Need For Speed World	*Online Game*
13	Midnight Club	Tamat
14	Test Drive	Belum Tamat
15	Test Drive Unlimited	Belum Tamat
16	Midnight Toon	Belum Tamat
17	Midnight Toon 3	Tamat
18	Midnight Toon 4	Belum Tamat
19	Midnight Toon 5	Tamat
20	Drift City	*Online Game*
21	Mario Kart Arcade GP	Belum Tamat

6.4 Pengujian AI DDA Melawan Pemain

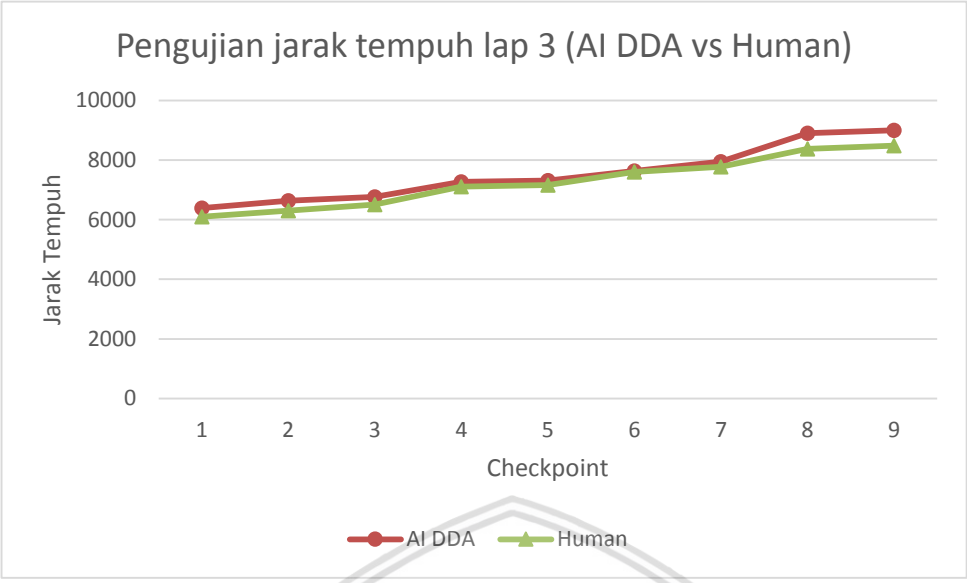
Pengujian dilakukan dengan cara membandingkan jarak tempuh antara AI dengan tingkat kesulitan yang dinamis (AI DDA) melawan pemain. Pengujian yang dilakukan sebanyak 3 *lap* dengan 9 *checkpoint* di setiap lap. Satuan jarak yang digunakan adalah satuan jarak pada Unity. Hasil pengujian jarak tempuh untuk AI DDA melawan pemain dapat dilihat pada Gambar 6.10, Gambar 6.11, dan Gambar 6.12.



Gambar 6.10 Grafik pengujian jarak tempuh lap 1 (AI DDA vs Pemain)



Gambar 6.11 Grafik pengujian jarak tempuh lap 2 (AI DDA vs Pemain)



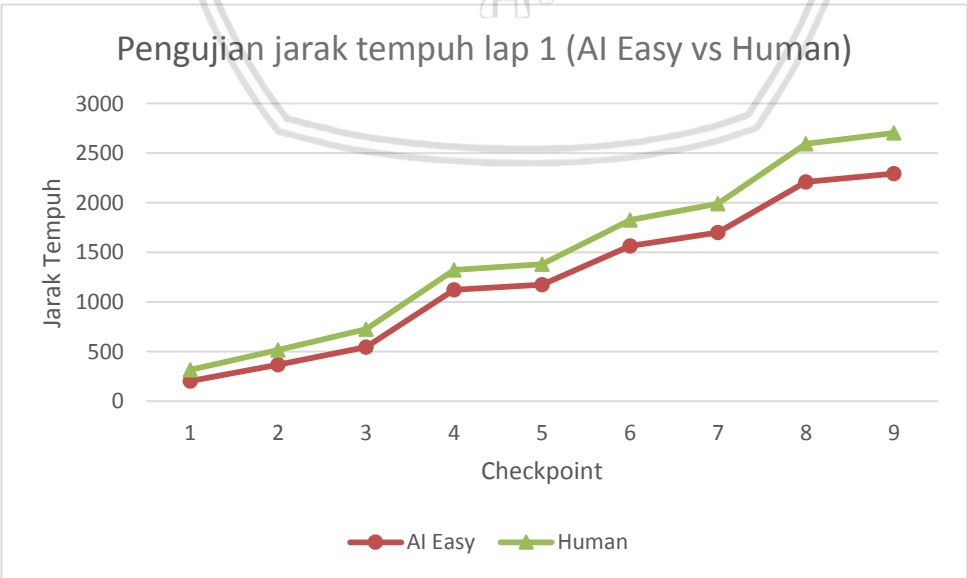
Gambar 6.12 Grafik pengujian jarak tempuh lap 3 (AI DDA vs Pemain)

6.5 Pengujian AI Statis Melawan Pemain

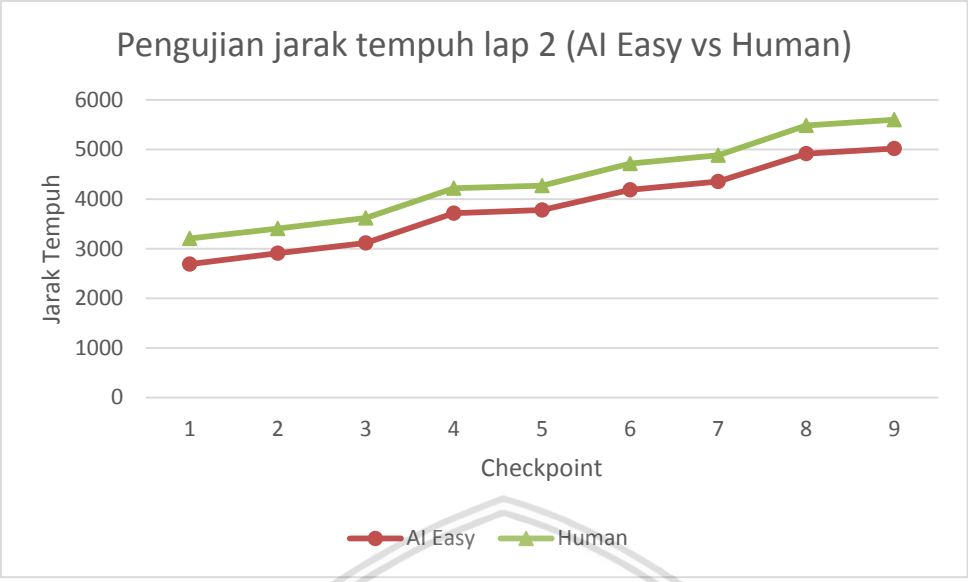
Pengujian dilakukan dengan cara membandingkan selisih jarak antara AI statis dengan tingkat kesulitan *easy, medium,* dan *hard* melawan pemain. Pada setiap skenario akan dilakukan pengujian sebanyak 3 *lap* dengan 9 *checkpoint* di setiap lap. Satuan jarak yang digunakan adalah satuan jarak pada unity.

6.5.1 Pengujian Jarak Tempuh (AI Easy vs Pemain)

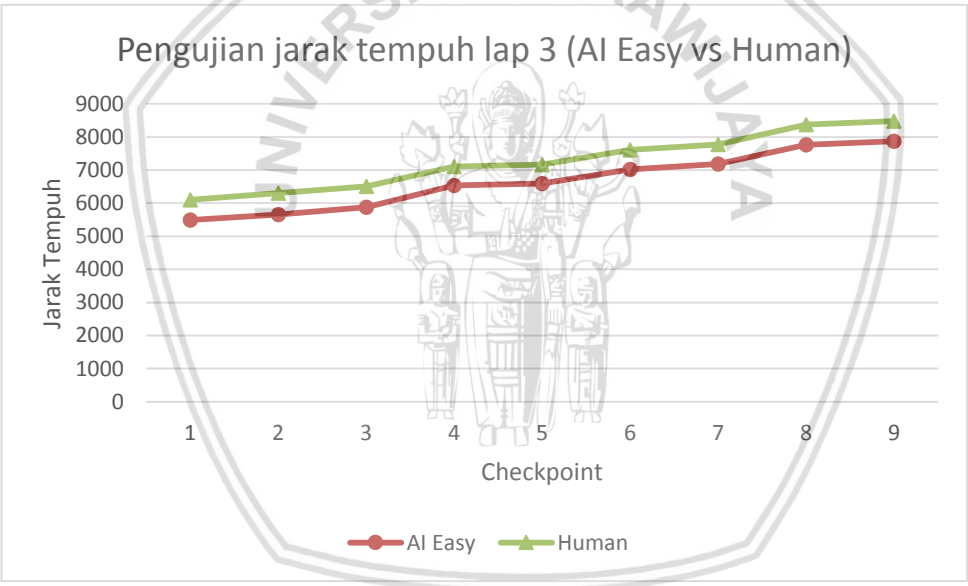
Hasil pengujian jarak tempuh untuk AI statis dengan tingkat kesulitan *easy* melawan pemain dapat dilihat pada Gambar 6.13, Gambar 6.14, dan Gambar 6.15.



Gambar 6.13 Grafik pengujian jarak tempuh lap 1 (AI Easy vs Pemain)



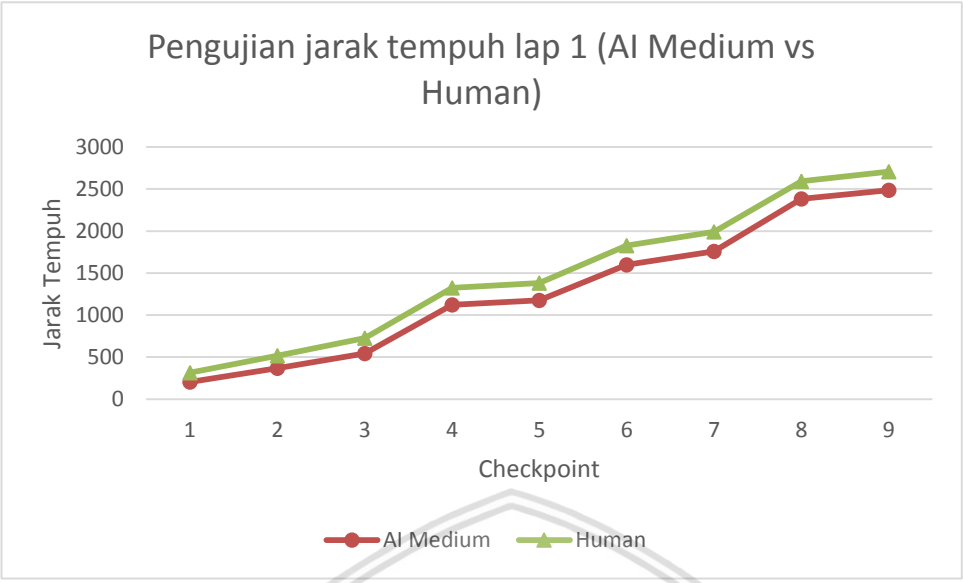
Gambar 6.14 Grafik pengujian jarak tempuh lap 2 (AI *Easy* vs Pemain)



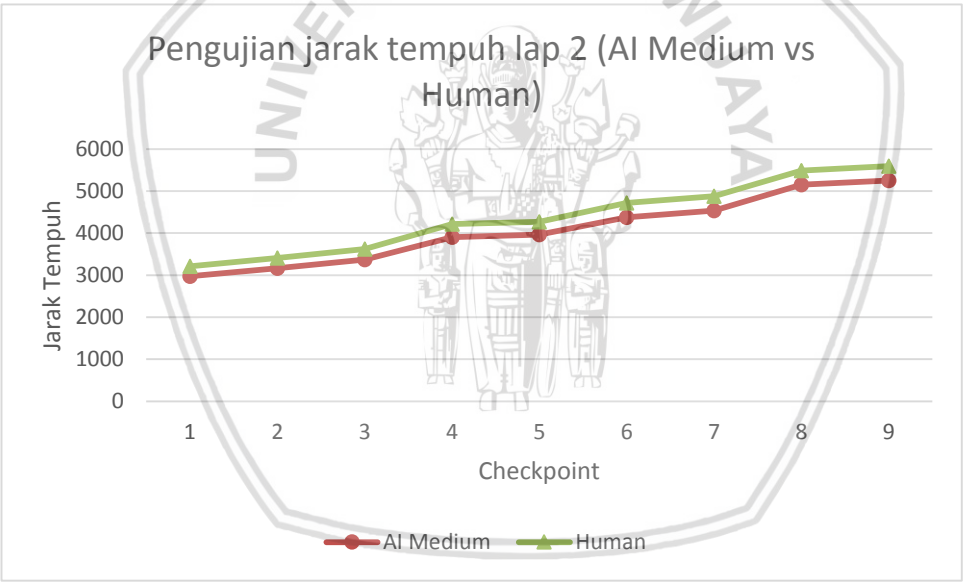
Gambar 6.15 Grafik pengujian jarak tempuh lap 3 (AI *Easy* vs Pemain)

6.5.2 Pengujian Jarak Tempuh (AI *Medium* vs Pemain)

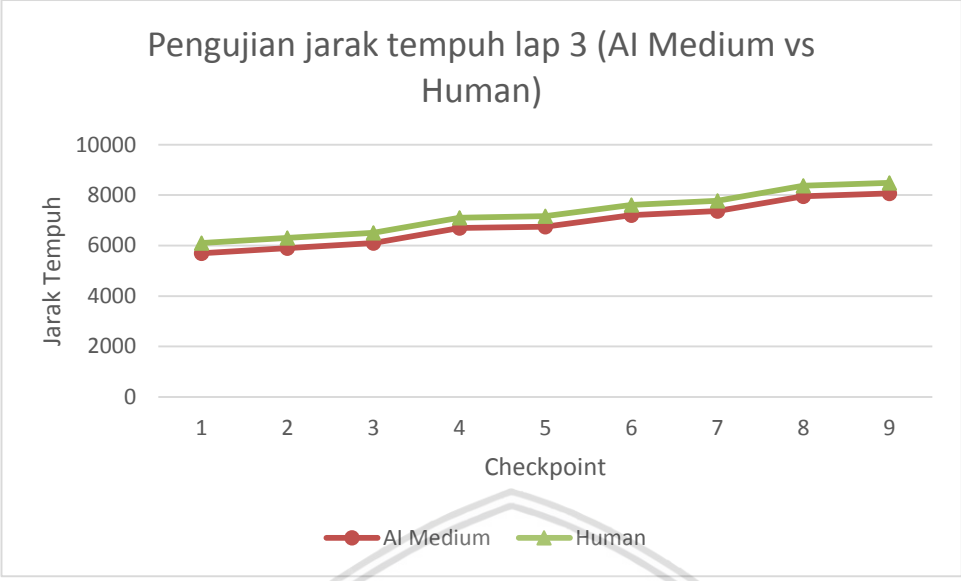
Hasil pengujian jarak tempuh untuk AI statis dengan tingkat kesulitan *medium* melawan pemain dapat dilihat pada Gambar 6.16, Gambar 6.17, dan Gambar 6.18.



Gambar 6.17 Grafik pengujian jarak tempuh lap 1 (AI *Medium* vs Pemain)



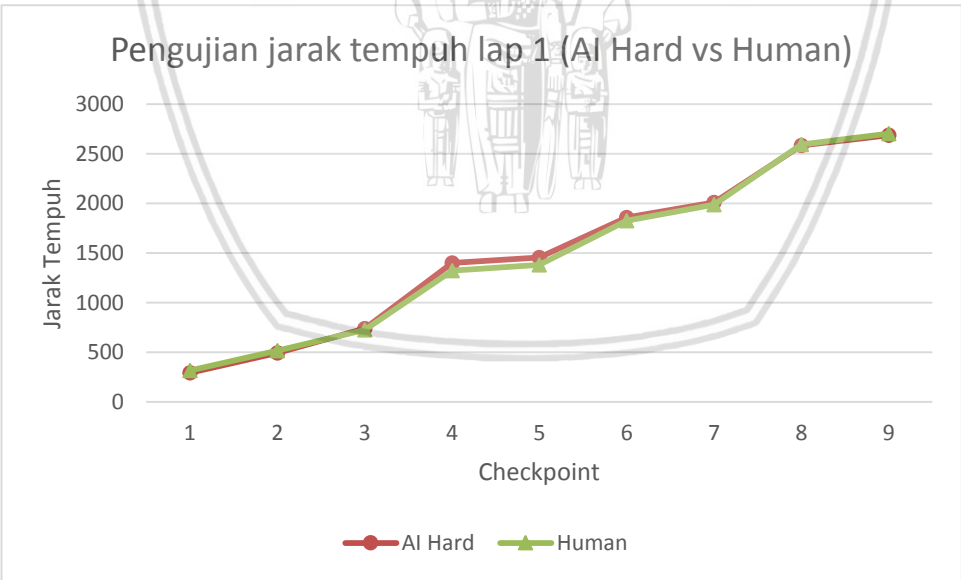
Gambar 6.16 Grafik pengujian jarak tempuh lap 2 (AI *Medium* vs Pemain)



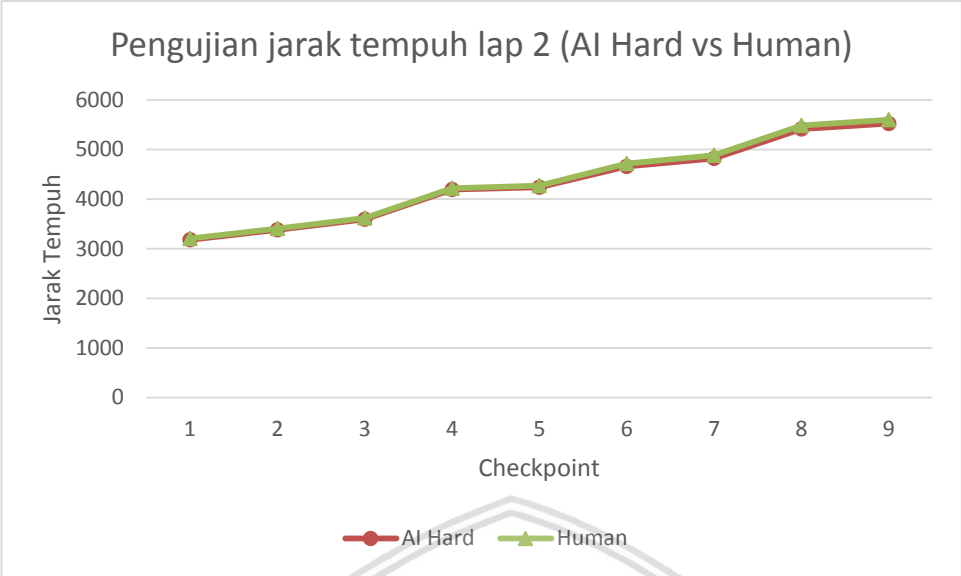
Gambar 6.18 Grafik pengujian jarak tempuh lap 3 (AI *Medium* vs Pemain)

6.5.3 Pengujian Jarak Tempuh (AI *Hard* vs Pemain)

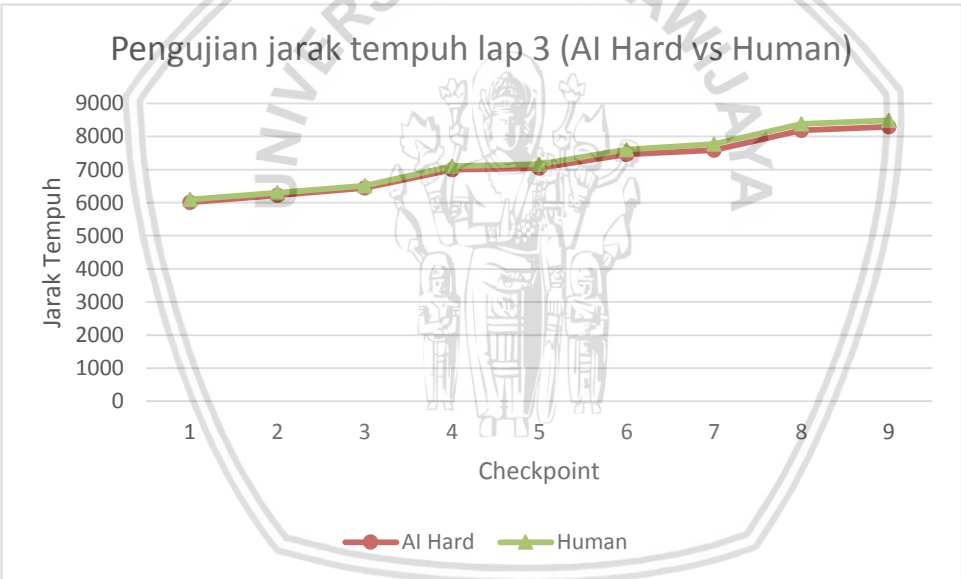
Hasil pengujian jarak tempuh untuk AI statis dengan tingkat kesulitan *hard* melawan pemain dapat dilihat pada Gambar 6.19, 6.20, 6.21.



Gambar 6.19 Grafik pengujian jarak tempuh lap 1 (AI *Hard* vs Pemain)



Gambar 6.20 Grafik pengujian jarak tempuh lap 2 (AI *Hard* vs Pemain)



Gambar 6.21 Grafik pengujian jarak tempuh lap 3 (AI *Hard* vs Pemain)

6.6 Analisis Pengujian AI DDA Melawan AI Statis

Terdapat 3 skenario pengujian yang dilakukan yaitu AI DDA melawan AI statis dengan tingkat kesulitan *easy*, AI DDA melawan AI statis dengan tingkat kesulitan m*edium*, dan AI DDA melawan AI statis dengan tingkat kesulitan *hard*.

6.6.1 Analisis Pengujian (AI DDA vs AI *Easy*)

Dapat dilihat pada Gambar 6.1 di *lap* 1 AI DDA berada pada posisi depan dengan selisih jarak yang besar dikarenakan DDA masih belum berjalan, namun setelah melewati *checkpoint* 1, AI DDA melambat karena menyusuaikan dengan kemampuan AI *easy* yang menjadi lawannya. Pada *lap* 2 dan 3 dapat dilihat pada Gambar 6.2, dan Gambar 6.3 AI *easy* berhasil meminimalkan selisih jarak sehingga posisi AI DDA dan AI easy berdempetan hingga *game* berakhr. Hasil ini menunjukkan bahwa AI DDA efektif terhadap lawan dengan kemampuan rendah.

6.6.2 Analisis Pengujian (AI DDA vs AI *Medium*)

Dapat dilihat pada Gambar 6.4 di *lap* 1 AI *medium* berada pada posisi depan, kemudian setelah melewati *checkpoint* 2, AI DDA membalap AI *medium* dan merebut posisi depan karena menyesuaikan dengan kemampuan AI *medium*. Pada *lap* 2 dapat dilihat pada Gambar 6.5, pada *checkpoint* 4 AI *medium* berhasil membalap AI DDA dikarenakan selisih jarak antara AI DDA dan AI *medium* terlalu besar sehingga membuat AI DDA menurunkan kemampuannya. Adanya saling balap antara AI DDA dan AI *medium* menunjukkan bahwa AI DDA efektif terhadap lawan dengan kemampuan sedang.

6.6.3 Analisis Pengujian (AI DDA vs AI *Hard*)

Dapat dilihat pada Gambar 6.7 di *lap* 1 AI *hard* berada pada posisi depan dengan selisih jarak yang besar hingga *lap* 1 berakhir. Pada *lap* 2 dapat dilihat pada Gambar 6.8, pada *checkpoint* 1 AI DDA berhasil mengejar ketinggalan dan meminimalkan selisish jarak dengan AI *hard* hingga posisi AI DDA dan AI *hard* berdempetan. Pada *lap* 3 dapat dilihat pada gambar 6.9, pada *checkpoint* 3 AI *hard* kembali meninggalkan AI DDA dengan selisih jarak yang cukup besar hingga *game* berakhir. Hasil ini menunjukkan bahwa AI DDA kurang efektif terhadap lawan dengan kemampuan sempurna.

6.7 Analisis Pengujian AI DDA dan AI Statis Melawan Pemain

Terdapat 4 skenario pengujian yang dilakukan yaitu pemain melawan AI DDA, pemain melawan AI statis dengan tingkat kesulitan *easy*, pemain melawan AI statis dengan tingkat kesulitan m*edium*, dan pemain melawan AI statis dengan tingkat kesulitan *hard*.

6.7.1 Analisis Pengujian (Pemain vs AI DDA)

Dapat dilihat pada Gambar 6.10 di *lap* 1 pemain berada pada posisi depan, kemudian setelah melewati *checkpoint* 3, AI DDA membalap pemain dan merebut posisi depan dengan menaikkan kemampuan AI DDA, namun pada *checkpoint* 5 pemain kembali merebut posisi depan dan memperbesar selisih jarak antara pemain dan AI DDA. Pada *lap* 2 dapat dilihat pada Gambar 6.11, pada *checkpoint* 9 AI DDA berhasil meminimalkan selisih jarak dengan pemain sehingga posisi pemain dan AI DDA berdempetan. Pada *lap* 3 dapat dilihat pada gambar 6.12, pada *checkpoint* 1 AI DDA merebut posisi depan kemudian posisi depan direbut kembali

oleh pemain di *checkpoint* 6, namun di *checkpoint* 7 AI DDA merebut kembali posisi depan dan mempertahankannya hingga *game* berakhir. Adanya saling balap antara pemain dan AI DDA menunjukkan bahwa AI DDA efektif terhadap lawan pemain berkemampuan handal.

6.7.2 Analisis Pengujian (Pemain vs AI *Easy*)

Dapat dilihat pada Gambar 6.13, Gambar 6.14, dan Gambar 6.15 pemain selalu berada di posisi depan sedangkan AI *easy* tertinggal semakin jauh di setiap lap dikarenakan tingkah laku AI *easy* yang *monotone*. Hasil ini menunjukkan bahwa pemain handal melawan AI *easy* menunjukkan hasil yang membosankan.

6.7.3 Analisis Pengujian (Pemain vs AI *Medium*)

Dapat dilihat pada Gambar 6.16, Gambar 6.17, dan Gambar 6.18 sama seperti melawan AI *easy* pemain selalu berada di posisi depan sedangkan AI *medium* tertinggal semakin jauh di setiap lap dikarenakan tingkah laku AI *medium* yang *monotone*. Hasil ini menunjukkan bahwa pemain handal melawan AI *medium* menunjukkan hasil yang membosankan.

6.7.4 Analisis Pengujian (Pemain vs AI *Hard*)

Dapat dilihat pada Gambar 6.19, Gambar 6.20, dan Gambar 6.21 dapat dilihat pada *lap* 1 *checkpoint* 3 AI *hard* berhasil merebut posisi depan, namun di *checkpoint* 7 pemain merebut kembali posisi depan dan mempertahankannya hingga game berakhir. Hasil ini menunjukkan bahwa pemain handal melawan AI *hard* menunjukkan hasil yang membosankan.

BAB 7 KESIMPULAN DAN SARAN

7.1 Kesimpulan

Berdasarkan pengujian yang dilakukan terhadap implementasi DDA pada AI *racing game* dengan metode *Behaviour Tree*, maka dapat diambil kesimpulan sebagai berikut:

1. Penerapan *Dynamic Difficulty Adjustment* pada AI *racing game* menggunakan *behaviour tree* yang di *update* setiap *checkpoint* menghasilkan AI dinamis yang bisa beradaptasi sesuai kemampuan lawan menggunakan masukan selisih jarak pada *Behaviour Tree* dan menghasilkan keluaran untuk mengubah parameter atribut AI.
2. Dari hasil pengujian AI dinamis dan AI statis melawan pemain yang dilakukan, didapat bahwa AI statis bertingkah laku *monotone* dan tidak bisa menyesuaikan dengan kemampuan pemain, sedangkan AI dinamis bertingkah laku secara dinamis dan dapat mengimbangi kemampuan pemain dengan baik sehingga membuat permainan jadi menyenangkan.

7.2 Saran

Saran untuk mendapatkan hasil yang lebih baik dari penilitian ini, dilakukan beberapa perbaikan sebagai berikut:

1. Diharapkan pada penelitian selanjutnya, dapat menambah atribut pada AI DDA supaya dapat berjalan lebih baik.
2. Penambahan parameter untuk perbandingan AI DDA dengan AI statis pada pengujian, sehingga rasio perbandingan data uji yang didapatkan akan meningkat.

DAFTAR PUSTAKA

Alexander, S., Francisco, M. G., Matthew, J., Robert, M., & Norman, I. B. (2011). Parameterizing Behavior Trees.

Andrade, G., Santana, H., Ramalho, G., & Corruble, V. (2005). Extending Reinforcement Learning to Provide Dynamic Game Balancing.

Csikszentmihalyi, M. (1991). Flow: The Psychology of Optimal Experience.

Intense Games. (2015, Juni 30). *Racing Game Starter Kit.* Retrieved from assetstore.unity.com: https://assetstore.unity.com/packages/templates/racing-game-starter-kit-22615 [Diakses 17 Juni 2018]

Kenneth, S., Torkil, O., & Long, H. (2011). Dynamic Difficulty Adjustment Using Behaviour Trees. *Artificial Intelligence in Video Games*.

Meniku. (2016, Juni 02). *NPBehave - An event driven Behavior Tree Library for code based AIs in Unity*. Retrieved from unitylist.com: https://unitylist.com/p/3ob/NP-Behave [Diakses 21 Juli 2018]

Michele, C., & Petter, O. (2017). Behavior Trees in Robotics and AI.

Mirna, P., Victor, d., & Luiz, C. (2016). Dynamic Difficulty Adjustment on MOBA Games. *Entertainment Computing*.

Rabin, S. (2008). AI Game Programming Wisdom 4. First Edition.

Robin, H., & Vernell, C. (2004). AI for Dynamic Difficulty Adjustment in Games.

Simon, T., & Nic, M. (2013). An Architecture Overview for AI in Racing Games. In T. Simon, & M. Nic, *Game AI Pro*.